This Is What We Find In This Stuff: A Designer Engineer's View

FY2005 Software/Complex Electronic Hardware Standardization Conference

Norfolk, Virginia
July 26-28, 2005

Rich Katz, Grunt Engineer
NASA Office of Logic Design

DO-254: Introduction

DESIGN ASSURANCE GUIDANCE FOR AIRBORNE ELECTRONIC HARDWARE

The use of increasingly complex electronic hardware for more of the ***safety critical aircraft functions*** *generates new safety and certification challenges. These challenges arise from a concern that said aircraft functions may be increasingly vulnerable to the adverse effects of* ***hardware design errors*** *that may be increasingly difficult to manage due to the increasing complexity of the hardware.*

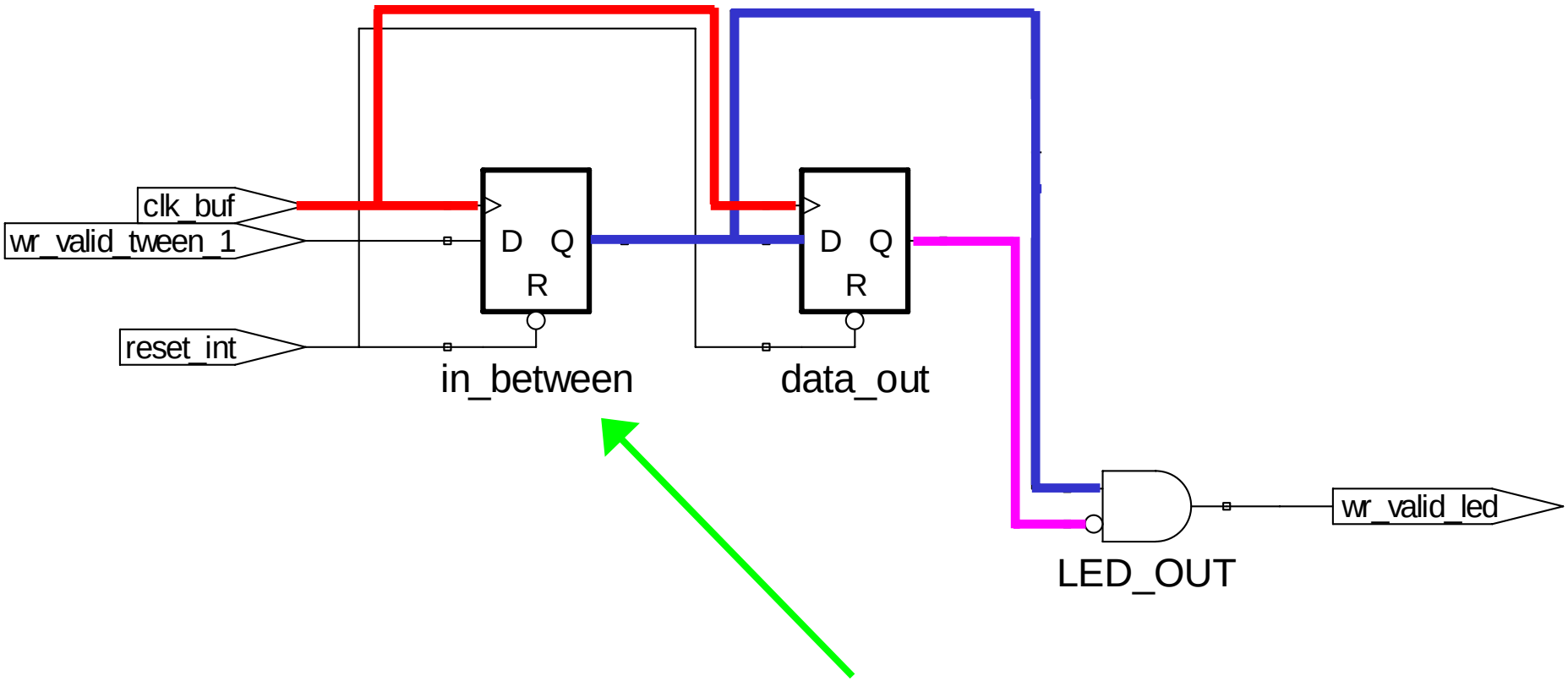
To counteract this ***perceived escalation of risk*** *it has become necessary to ensure that the potential for hardware design errors is addressed in a more consistent and verifiable manner during both the design and certification processes.*

An Example or Two to Get Things Started

CAE Tools and Logic Design: Synchronization Circuits

Intended Circuit

Synchronizer with Leading Edge Detect

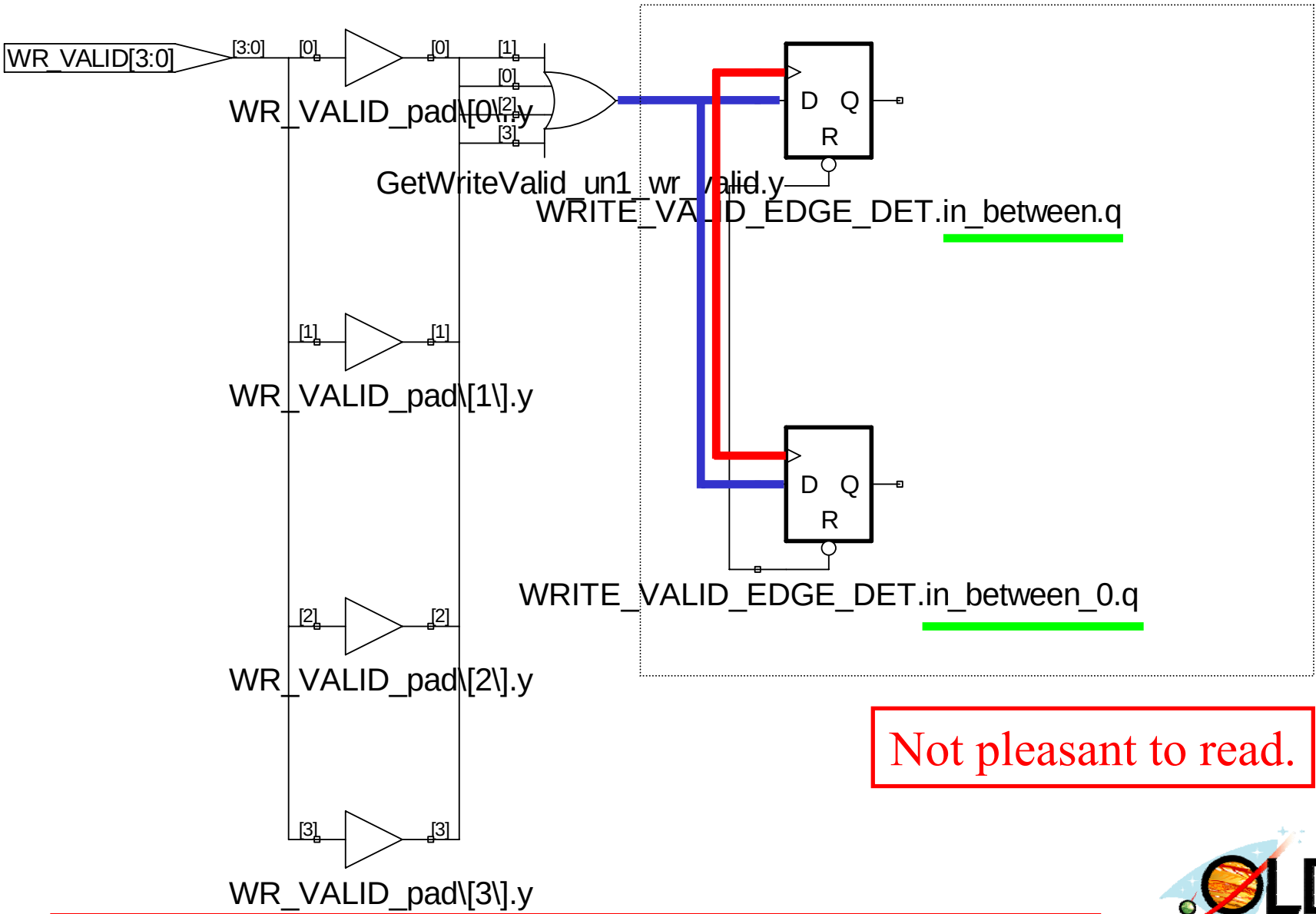


VHDL Code

```
entity EDGE_DETECT_SYNC is
   generic (
      RESET_LEVEL : STD_LOGIC := '1' );

   port (
      CLK         : in  STD_LOGIC;
      RESET       : in  STD_LOGIC;
      INPUT       : in  STD_LOGIC;
      LED_OUT     : out STD_LOGIC;
      TED_OUT     : out STD_LOGIC
    );
end EDGE_DETECT_SYNC;

architecture BEHAVIORAL of EDGE_DETECT_SYNC
is

   signal IN_BETWEEN : STD_LOGIC;
   signal DATA_OUT   : STD_LOGIC;
```

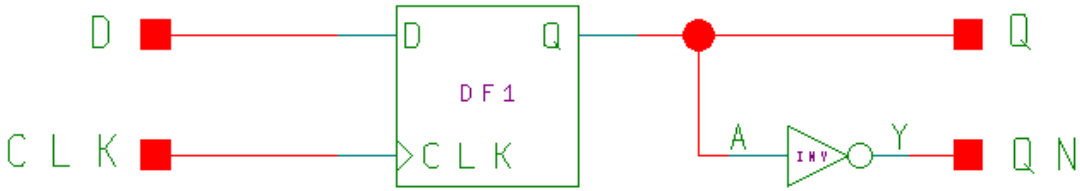
```
begin

FF1 : process(RESET, CLK)
   begin
      if (RESET = RESET_LEVEL) then
         IN_BETWEEN <= '0';
      elsif rising_edge(CLK) then
         IN_BETWEEN <= INPUT;
      end if;
   end process;


FF2 : process(RESET, CLK)
   begin
      if (RESET = RESET_LEVEL) then
         DATA_OUT <= '0';
      elsif rising_edge(CLK) then
         DATA_OUT <= IN_BETWEEN;
      end if;
   end process;

   LED_OUT <= (not DATA_OUT) and IN_BETWEEN;
   TED_OUT <=  DATA_OUT and (not IN_BETWEEN);

end BEHAVIORAL;
```

Synthesized Circuit

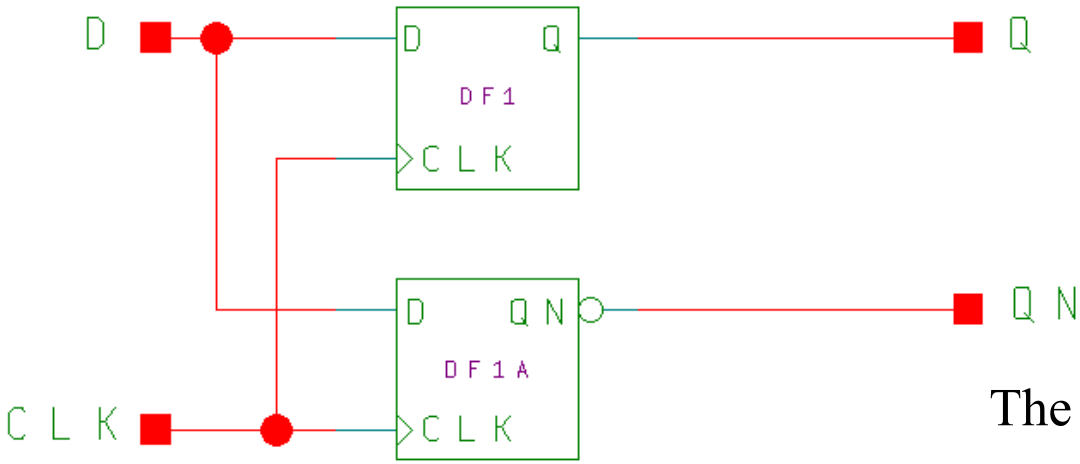


Not pleasant to read.

Synthesized Circuit



Original



"Optimized"

The designer used this point to synchronize signals and drive a motor. The short circuit was "bad."

Back end tools also have been caught replicating flip-flops in synchronizer circuits.

Dr. Wernher von Braun
Director Marshall Space Flight Center
Ph.D. Aerospace Engineering

James E. Webb, NASA Administrator
BA Education, Lawyer

Basic Digital Logic and Programmable Devices

Basic Digital Operations

A	Y
0	1
1	0

Not
'

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

AND
.

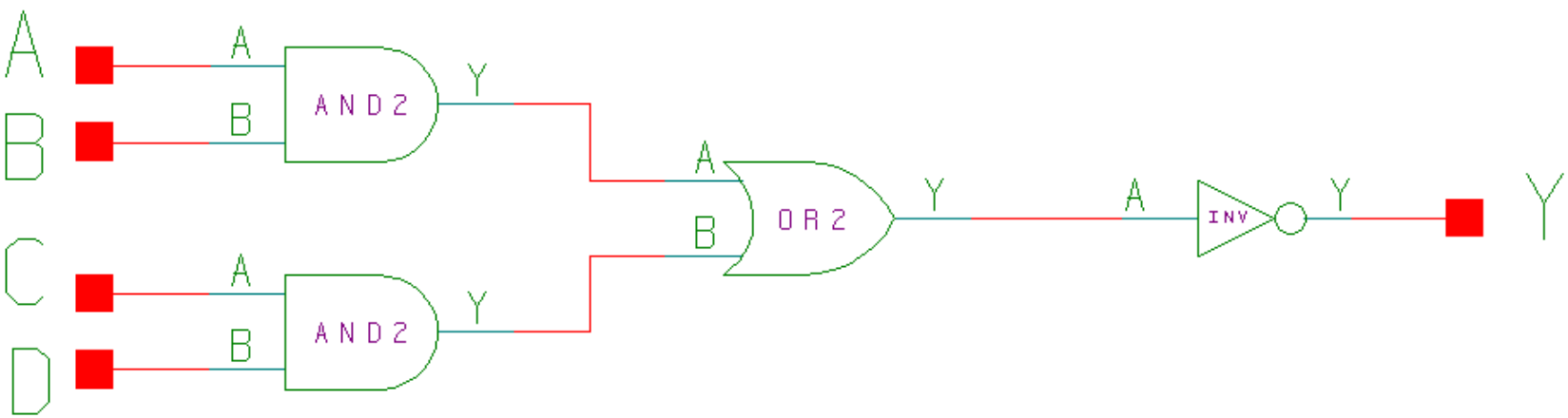
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

OR
+

There are $2^4 = 16$ functions of two binary variables.

Logic Functions

- Any logic function can be implemented with AND, OR, and NOT.
- One standard form is the sum of products
  - Example: $Y = (A \bullet B + C \bullet D)'$

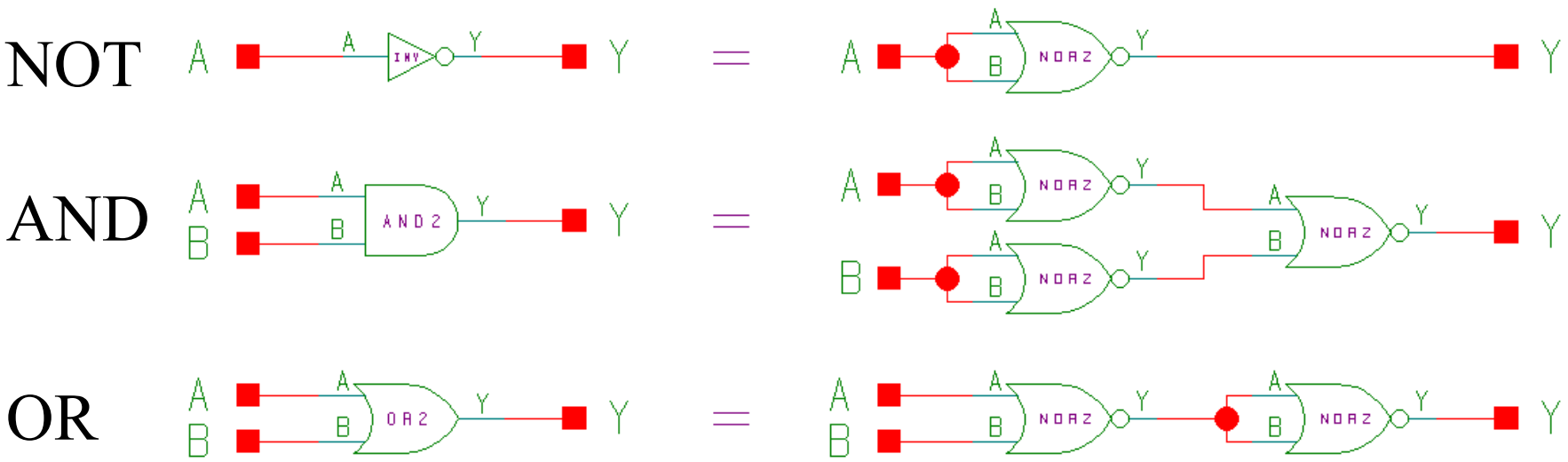


Inputs **AND gates** **OR gates** **Inverters** **Outputs**

Universal Logic Gate
NOR Function

- NOR ::= Negative OR

$$Y = (A + B)'$$

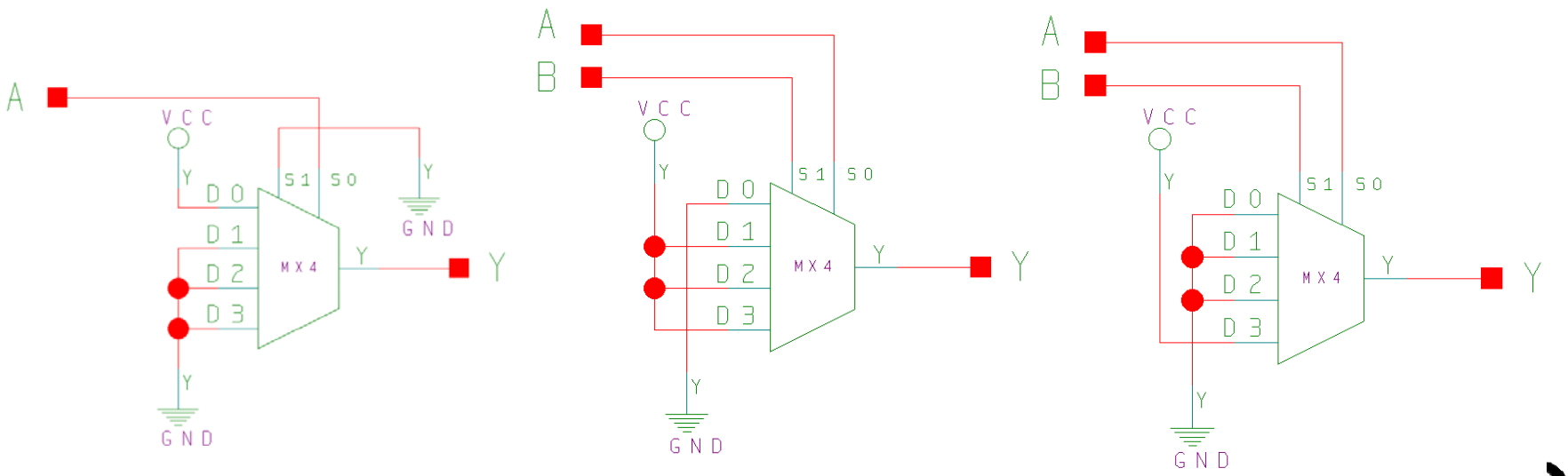


Can perform similar operation with NAND gate.

Universal Logic Gate

Multiplexor: $Y = A \bullet S + B \bullet S'$

NOT **OR** **AND**



Universal Logic Gate Look up table (LUT)

- Look up table (LUT)

  Small memory

NOT

A	X	Y
0	0	1
0	1	1
1	0	0
1	1	0

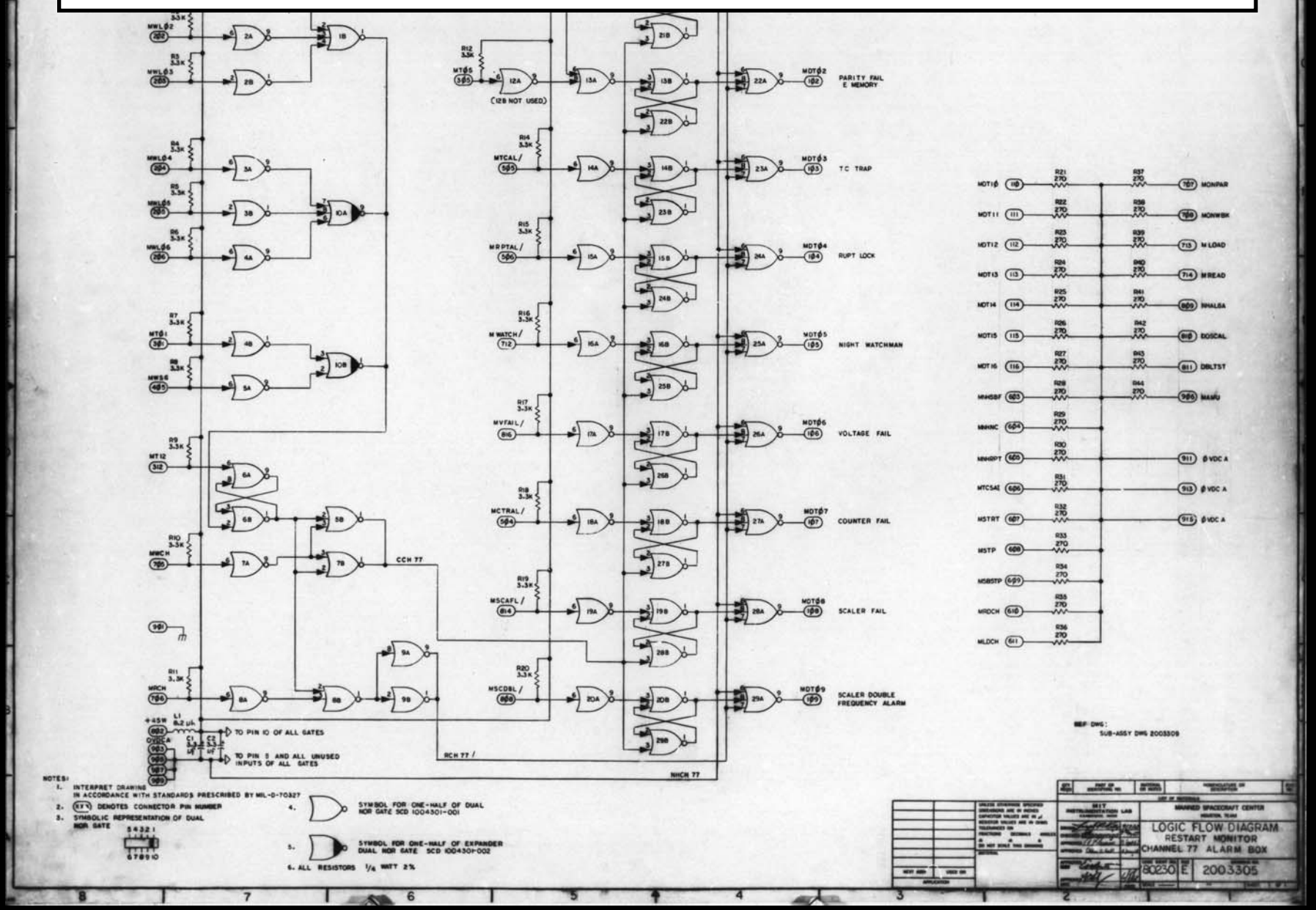
OR

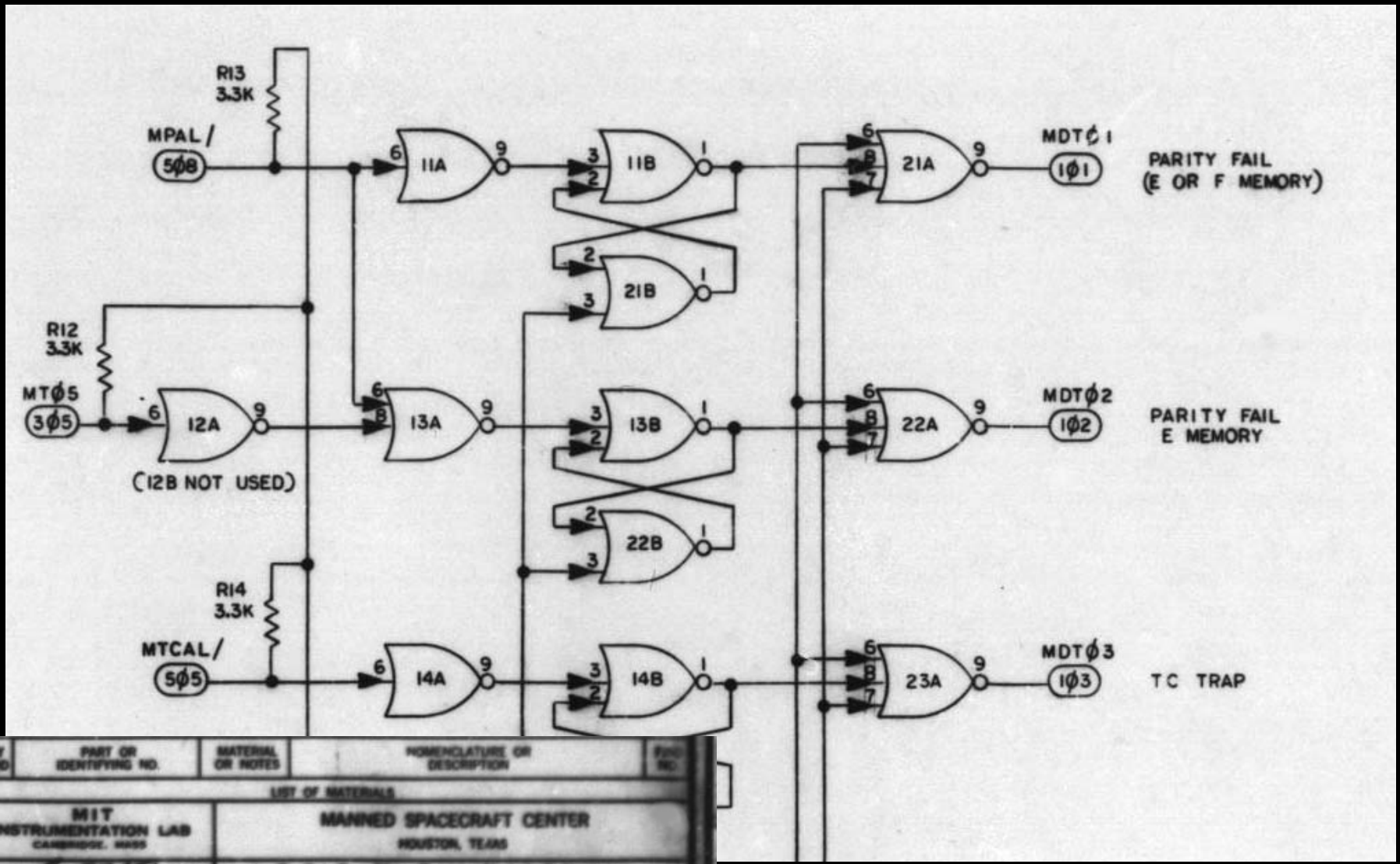
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

A Practical Example of the Universal Logic Element





QTY REQD	PART OR IDENTIFYING NO.	MATERIAL OR NOTES	NOMENCLATURE OR DESCRIPTION	FIND NO.

LIST OF MATERIALS

MIT
INSTRUMENTATION LAB
CAMBRIDGE, MASS

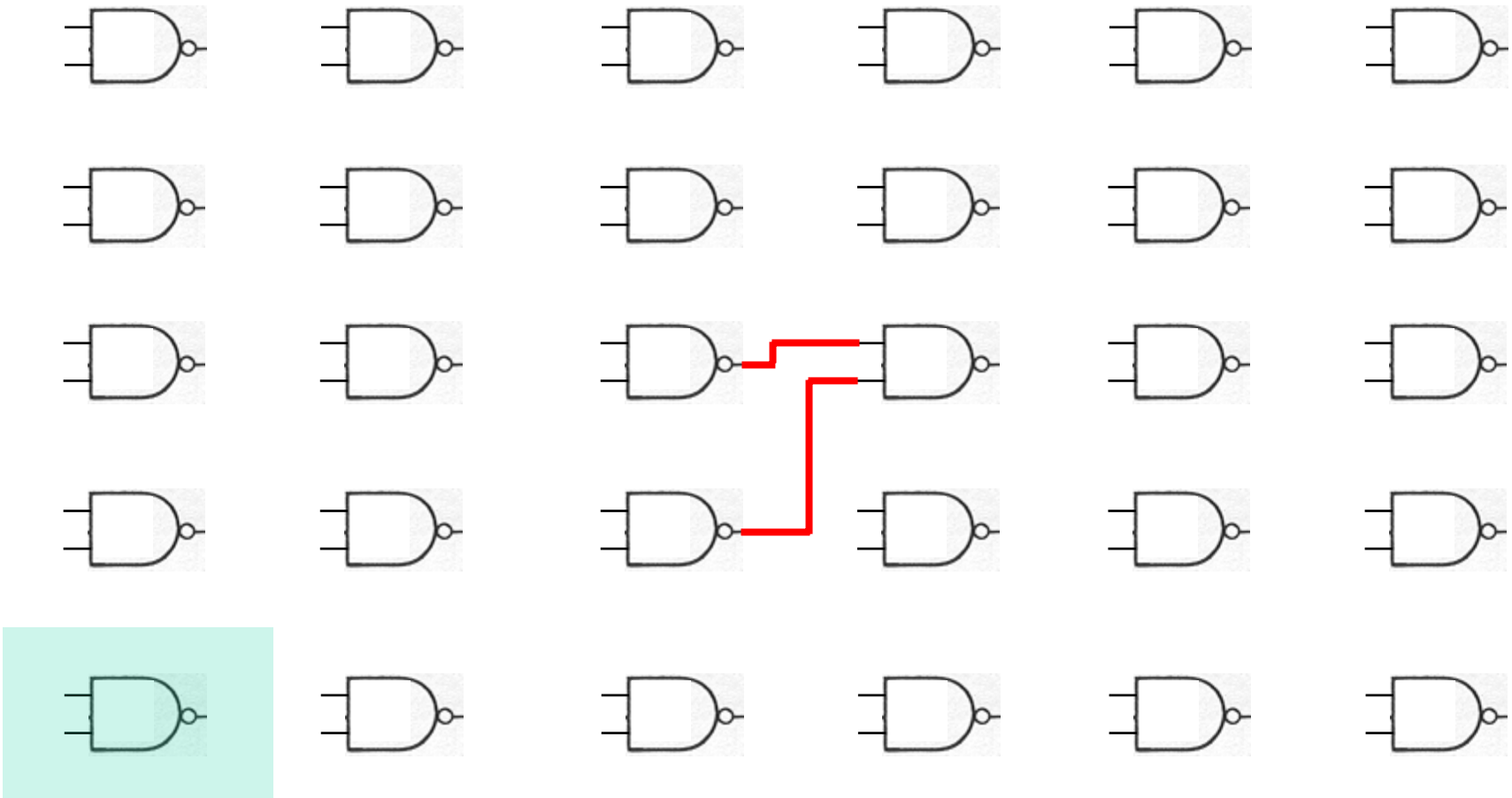
MANNED SPACECRAFT CENTER
HOUSTON, TEXAS

LOGIC FLOW DIAGRAM
RESTART MONITOR
CHANNEL 77 ALARM BOX

CODE IDENT NO.	SIZE	DRAWING NO.
80230	E	2003305

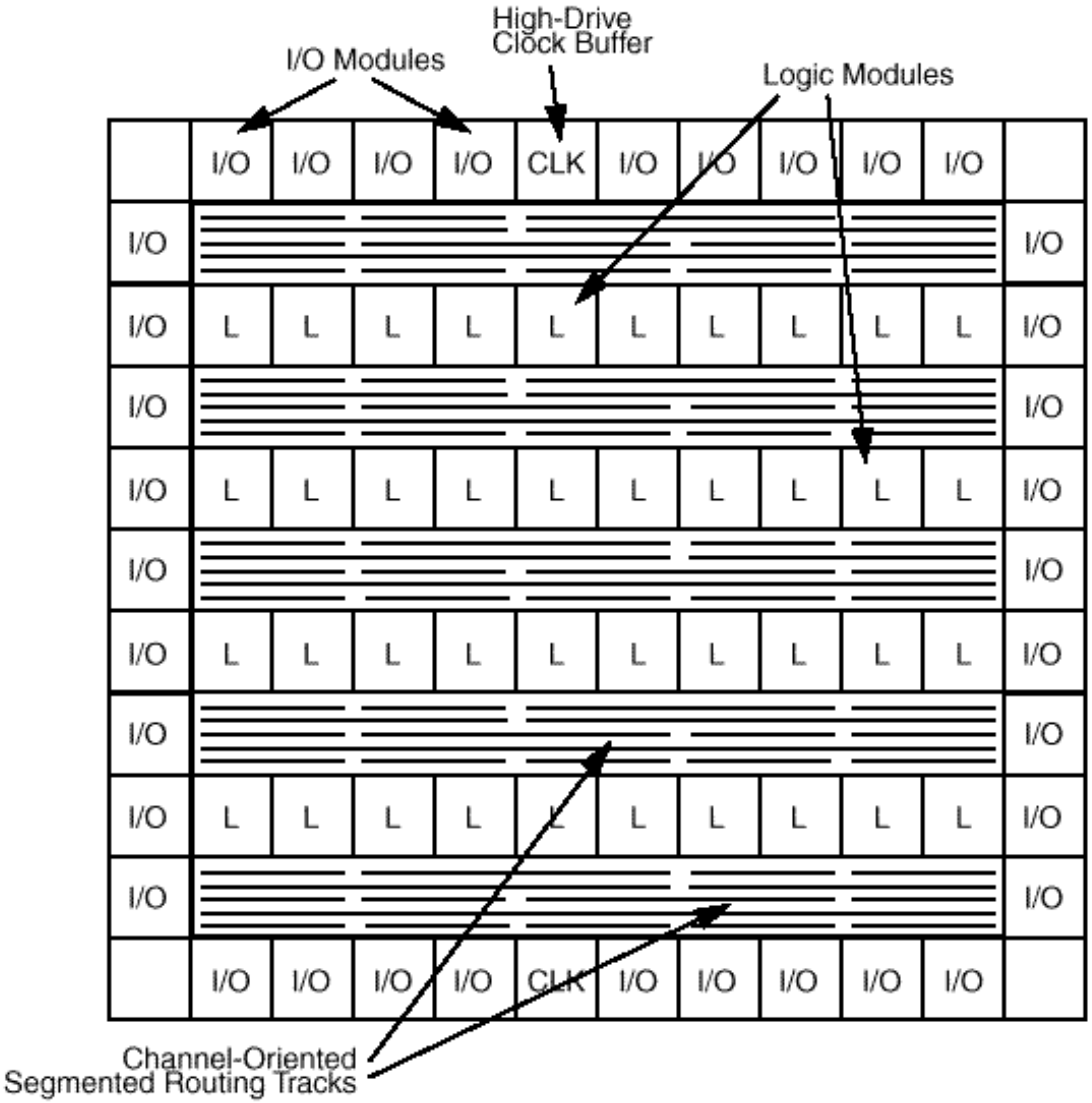
SCALE —— SHEET 1 OF 1

What Is A Gate Array?

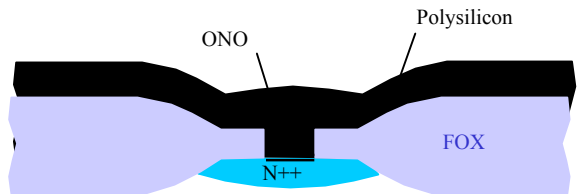


What Is An Antifuse-Based FPGA?

Antifuse-FPGA Architecture



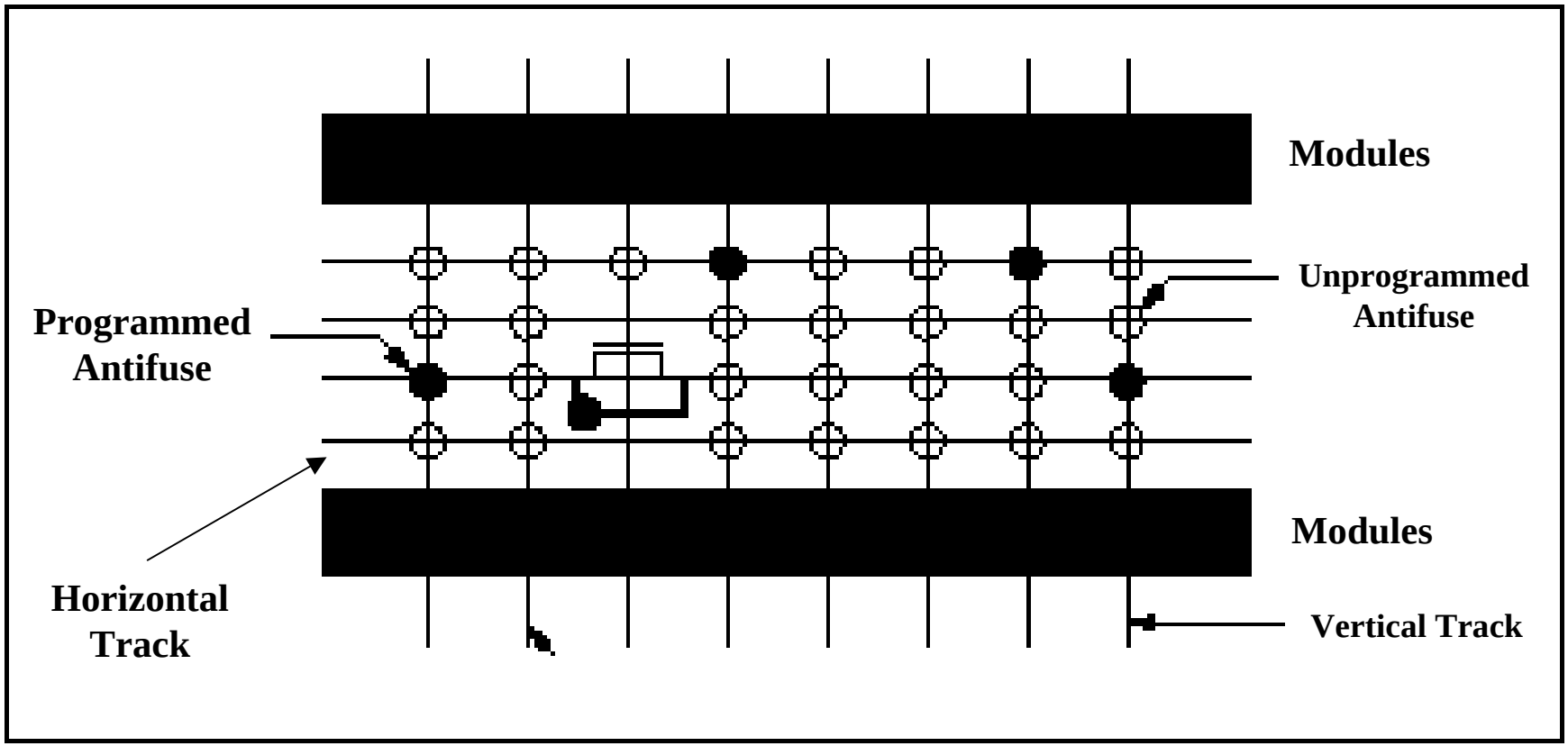
In some FPGAs, the routing resources and antifuses are above the logic modules in a sea-of-modules architecture. This is enabled by the metal-to-metal antifuse.



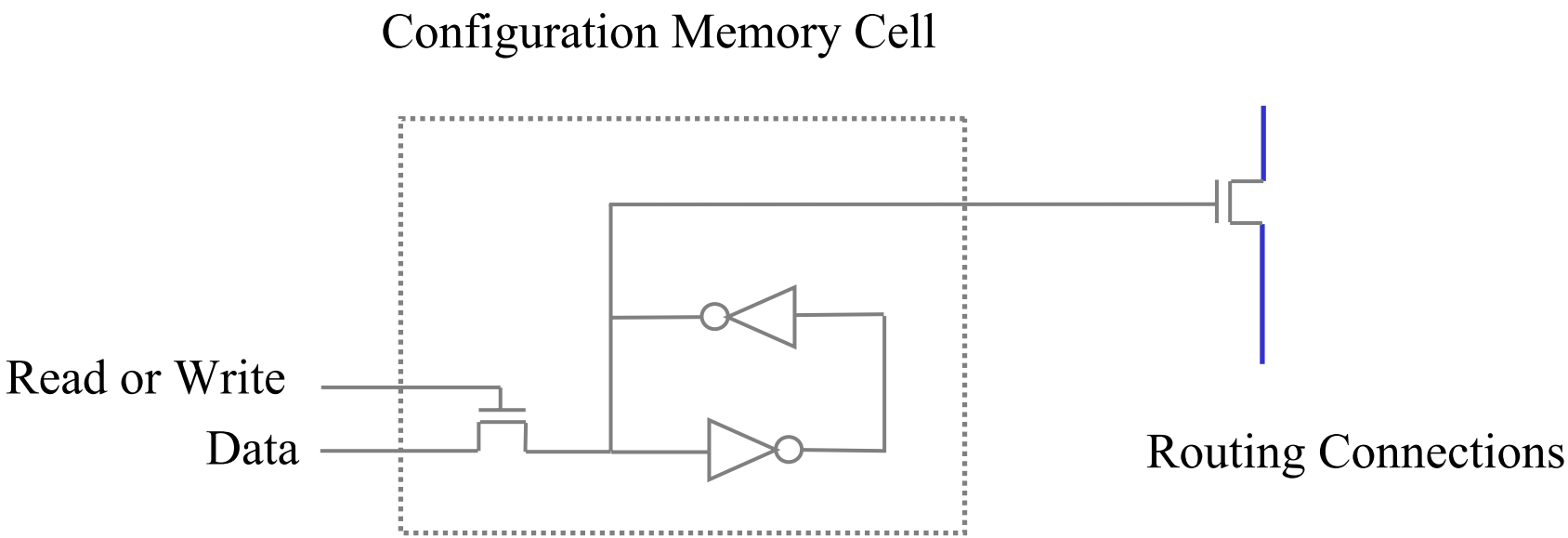
Antifuse resistance ranges from 200 to 500 Ω for ONO technology and about 25 Ω for metal-to-metal types.

What Is An Antifuse-Based FPGA?

Antifuse-FPGA Architecture



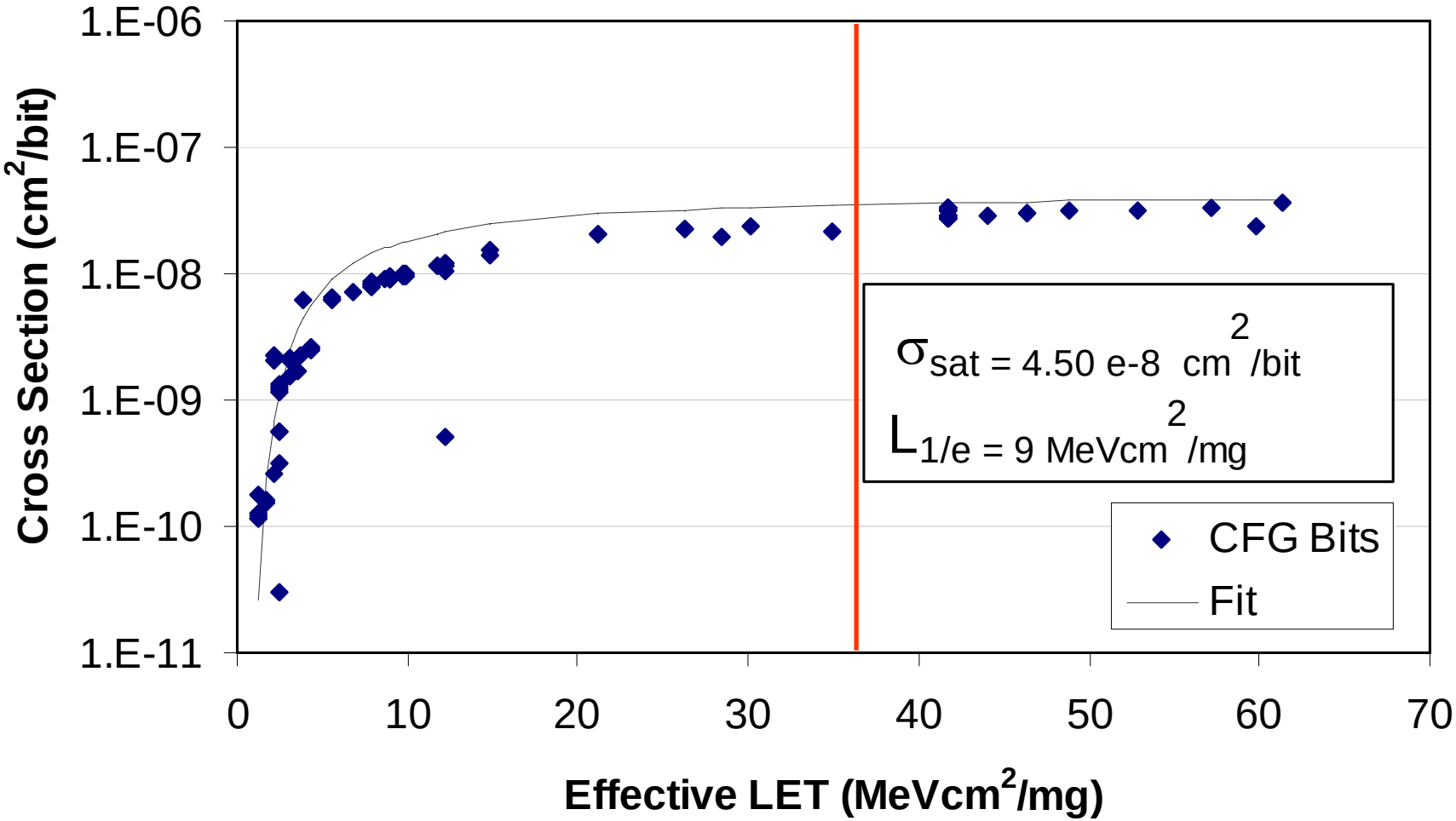
SRAM Switch Technology



> 10^7 configuration memory cells per device

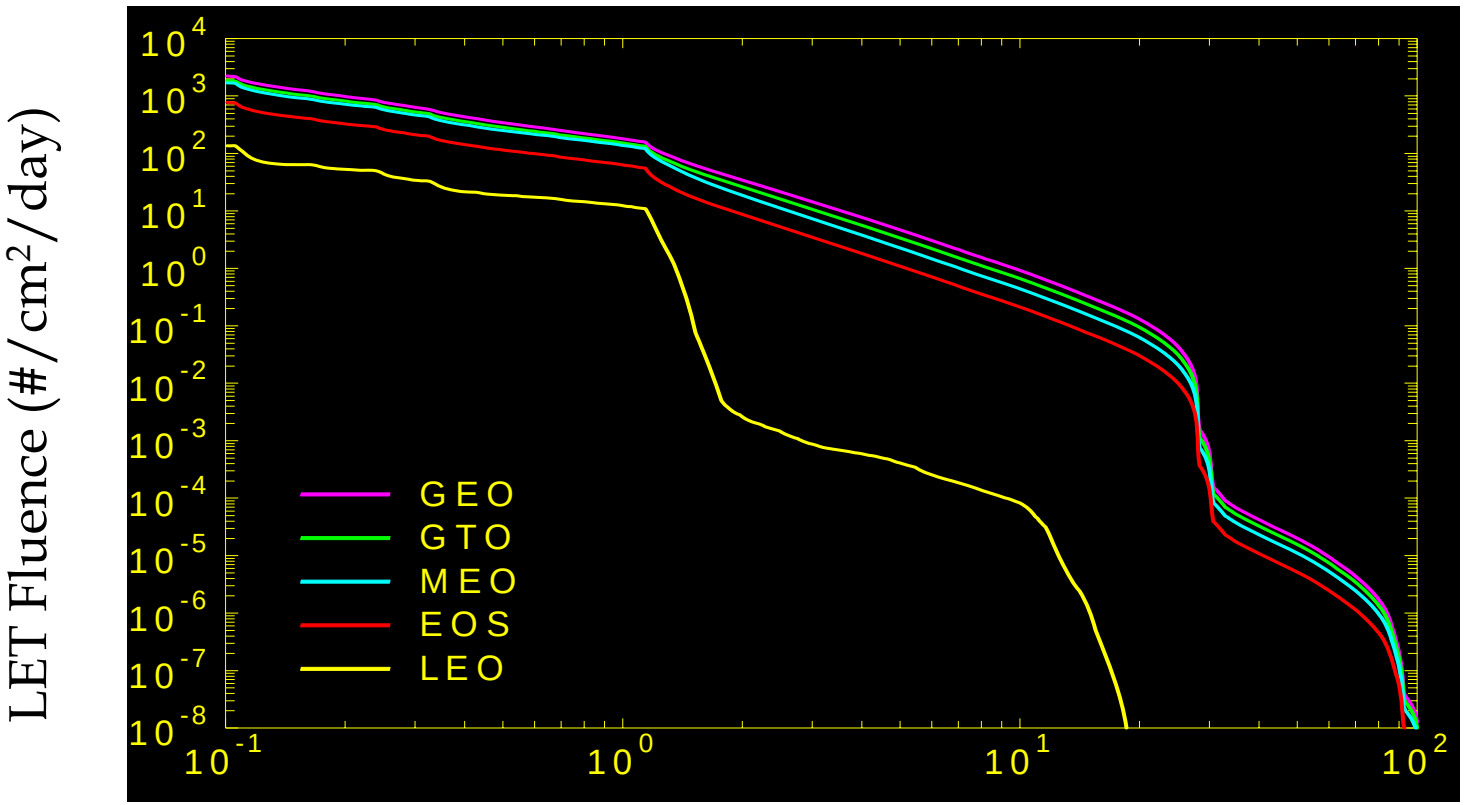
Virtex II XC2V1000 SEU Data

Configuration Bits



Galactic Cosmic Rays:
Integral LET Spectra

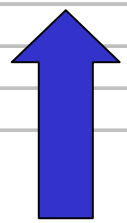
CREME 96, Solar Minimum, 100 mils (2.54 mm) Al



Virtex II 2V6000 SEU Data

Atmospheric Neutrons

Location	Days	Hours	Upsets	Device Hours	Bits	>10MeV Flux (n/cm2-hr)	>10MeV Fluence	>10MeV Cross-Section
San Jose	443	10632	6	1.06E+06	1.96E+09	14.40	153,101	2.00E-14
ABQ	564	13536	34	1.35E+06	1.96E+09	53.28	721,198	2.41E-14
WM	229	5496	66	5.50E+05	1.96E+09	338.40	1,859,846	1.81E-14
MK	90	2160	18	2.16E+05	1.96E+09	229.57	495,880	1.85E-14
							Average Cross-Section	2.02E-14
							LANSCE 2V6000	3.05E-14
							Rosetta Factor	**1.51**



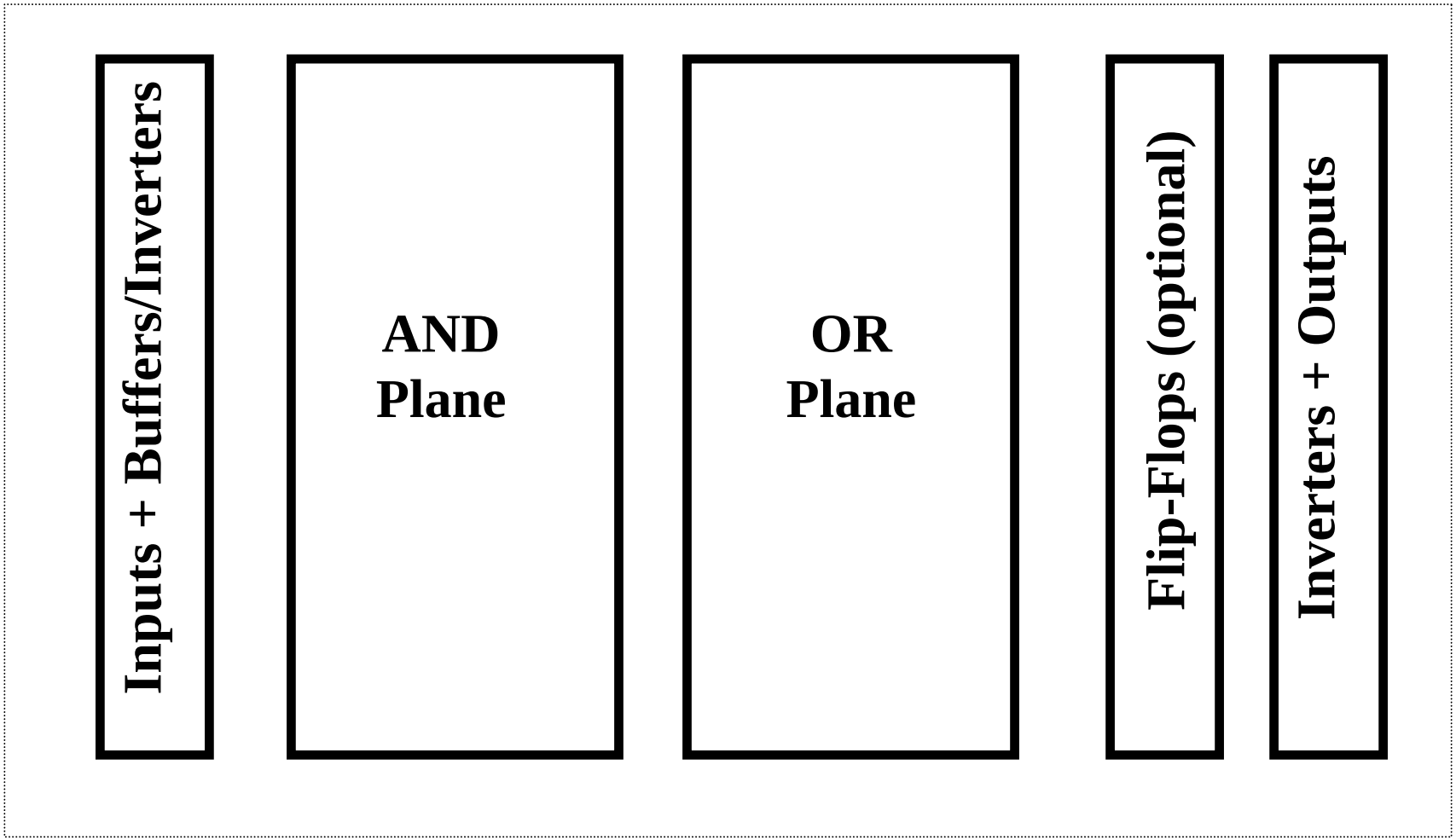
Test operating at 4 altitudes

- **Sea Level – San Jose**
- **5,200 feet – Albuquerque**
- **12,000 feet – White Mountain Research Center**
- **13,500 feet – Mauna Kea Observatory**

Module Design of

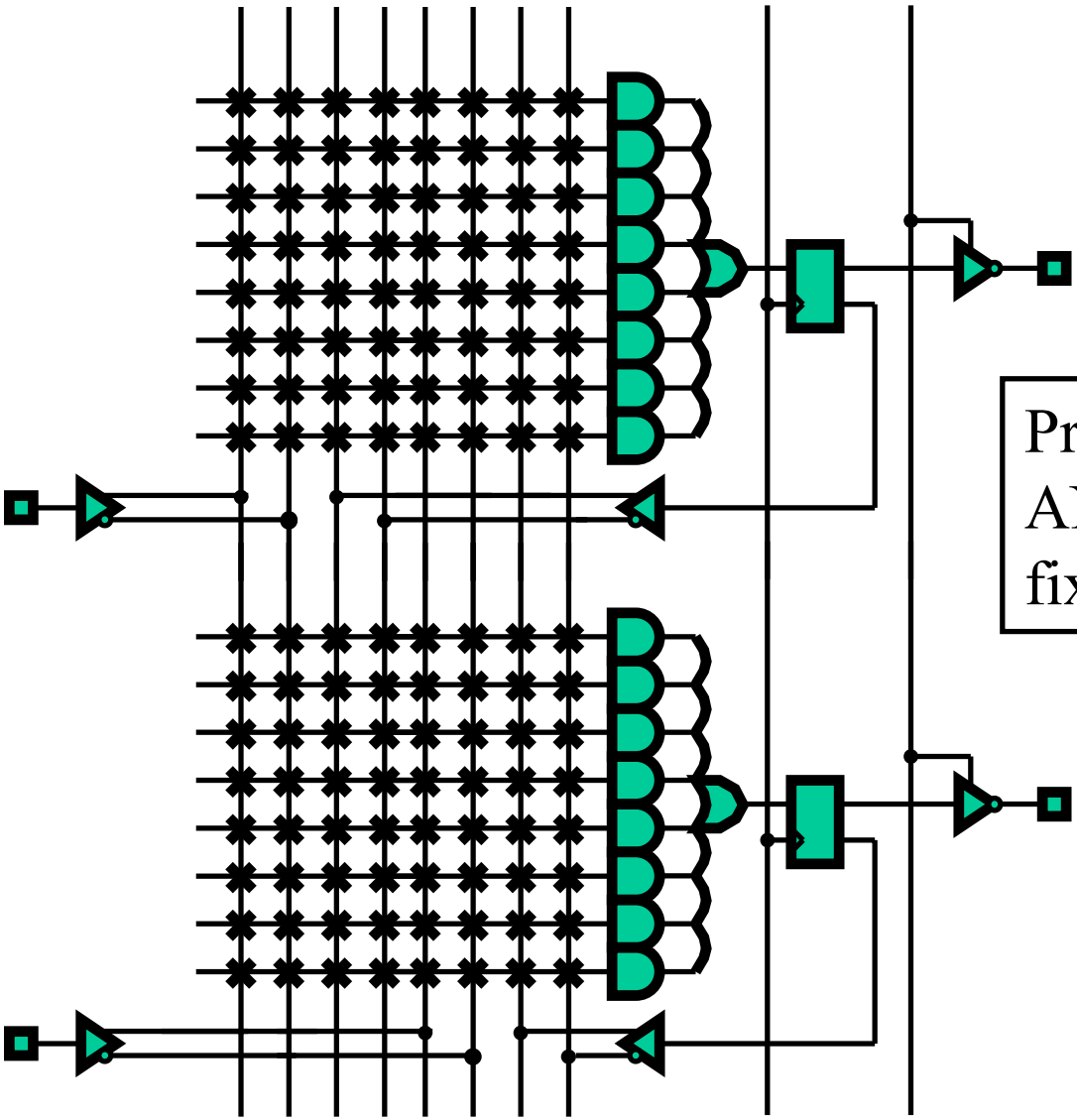
PALs and FPGAs

Programmable Logic Components



PROMs, PALs, and PLAs all have a similar architecture

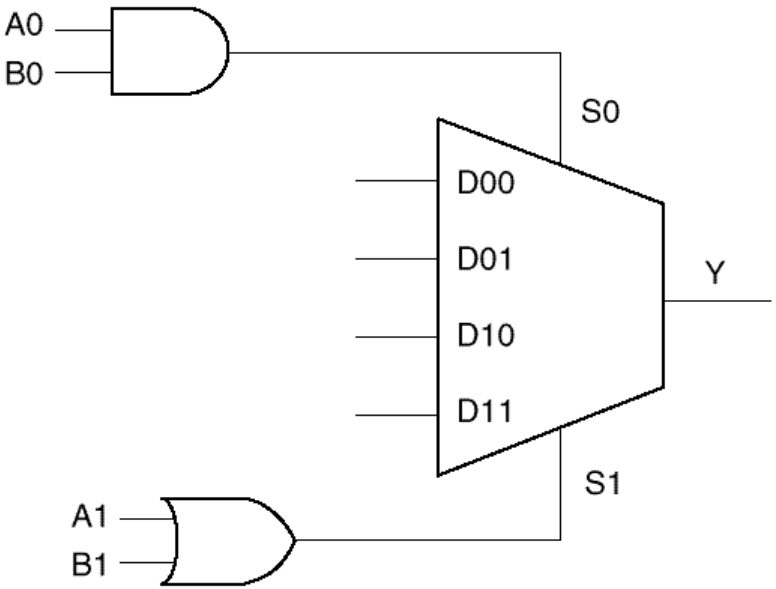
PAL Architecture

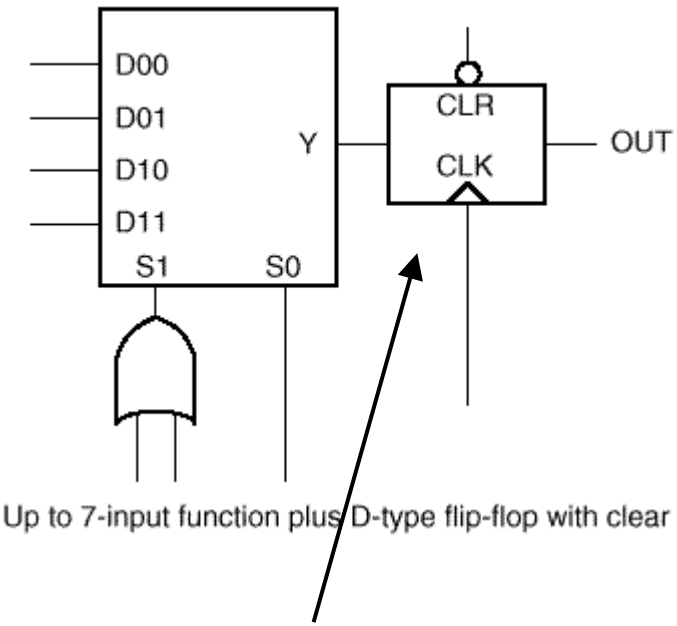


Programmable AND plane and fixed OR plane.

PALs have a built-in POR circuit to initialize all registers to zero.

Act 2 Logic Modules





8-Input Combinational function

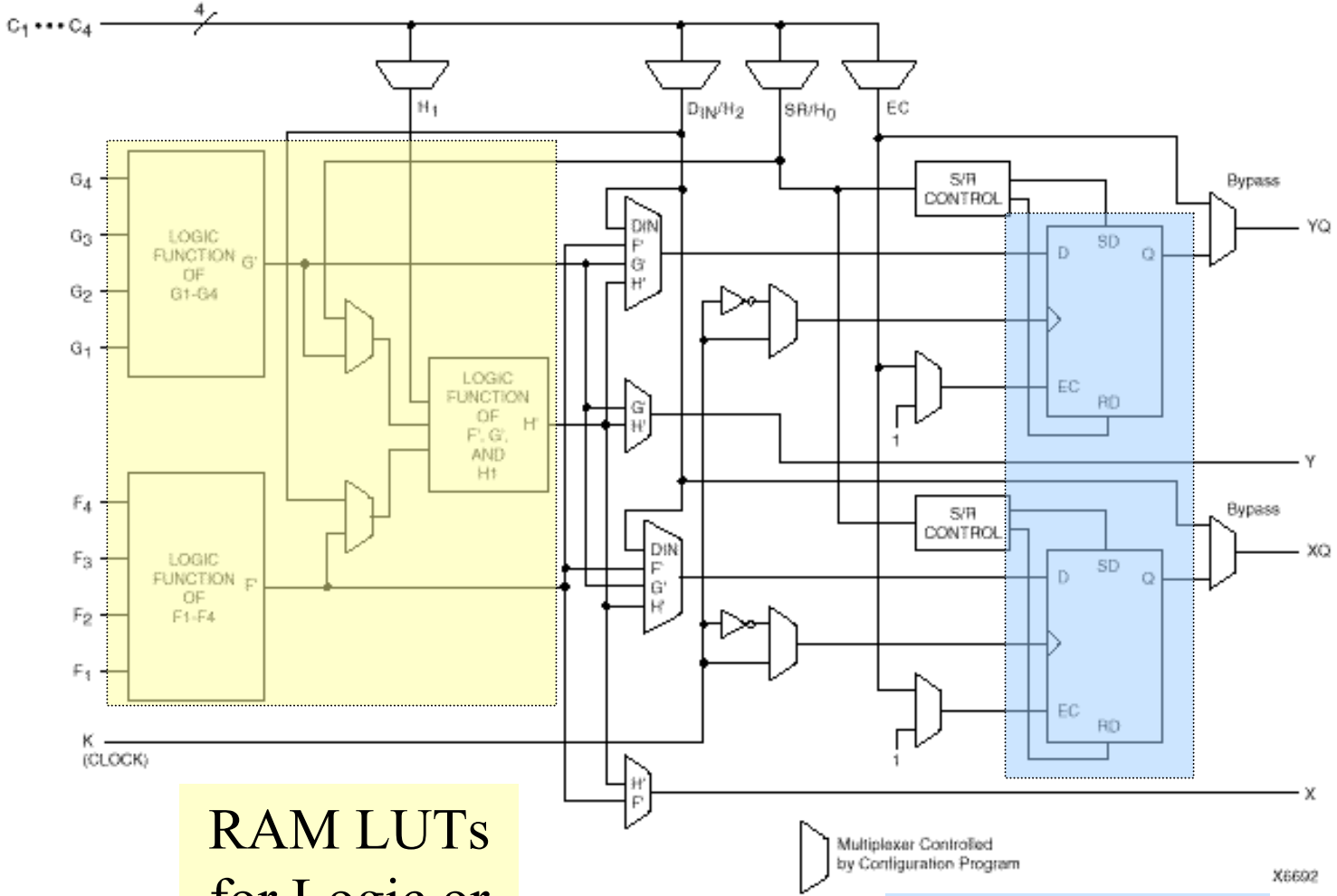
766 possible combinational macros[1]

Hard-wired Flip-flop

[1]"Antifuse Field Programmable Gate Arrays," J. Greene, E. Hamdy, and S. Beal, **Proceedings of the IEEE**, Vol. 91, No. 7, July 1993, pp. 1042-1056

XC4000 Series CLB

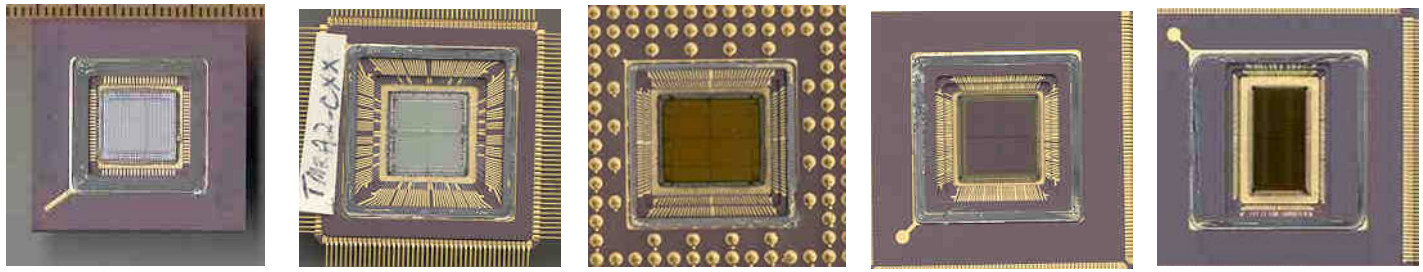
Simplified CLB - Carry Logic Not Shown



RAM LUTs for Logic or small SRAM

Two Flip-flops

Five Generations of Actel Space Flight FPGAs



Model	A1020	A1280A	A14100A	RTSX32	RTSX72S
Foundry	MEC	MEC	MEC	MEC	MEC
Feature Size	2.0 µm	1.0 µm	0.8 µm	0.6 µm	0.25 µm
Supply Voltages	5.0V	5.0V	5.0V	5.0V/3.3V	5.0V/2.5V
Gate Density	1x	2x	5x	8x	18x
Antifuse Type	ONO	ONO	ONO	M2M	M2M
					SEU-Hardened
Die Size	.363” x .387”	.416” x .433”	.533” x .437”	.428” x .412”	.430” x .675”

Notes:
- Largest device in each family shown
- RH parts available: RH1020, RH1280; TID hardened only
- Gate counts approximate

Some Definitions

DESIGN ASSURANCE GUIDANCE
FOR AIRBORNE ELECTRONIC HARDWARE

RTCA/DO-254

April 19, 2000

SC-180

Definitions

Programmable Logic Device (PLD) *- A component that is purchased as an electronic component and altered to perform an application specific function. PLDs include, but are not limited to, Programmable Array Logic components, Programmable Logic Array components, General Array Logic components, Field Programmable Gate Array components and Erasable Programmable Logic Devices.*

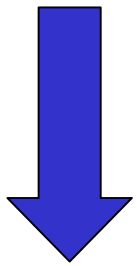
Design Tools *- Tools whose output is part of hardware design and thus can introduce errors. For example, an ASIC router or a tool that creates a board or chip layout based on a schematic or other detailed requirement.*

Design Assurance *– All of those planned and systematic actions used to substantiate, at an adequate level of confidence, that design errors have been identified and corrected such that the hardware satisfies the application certification basis.*

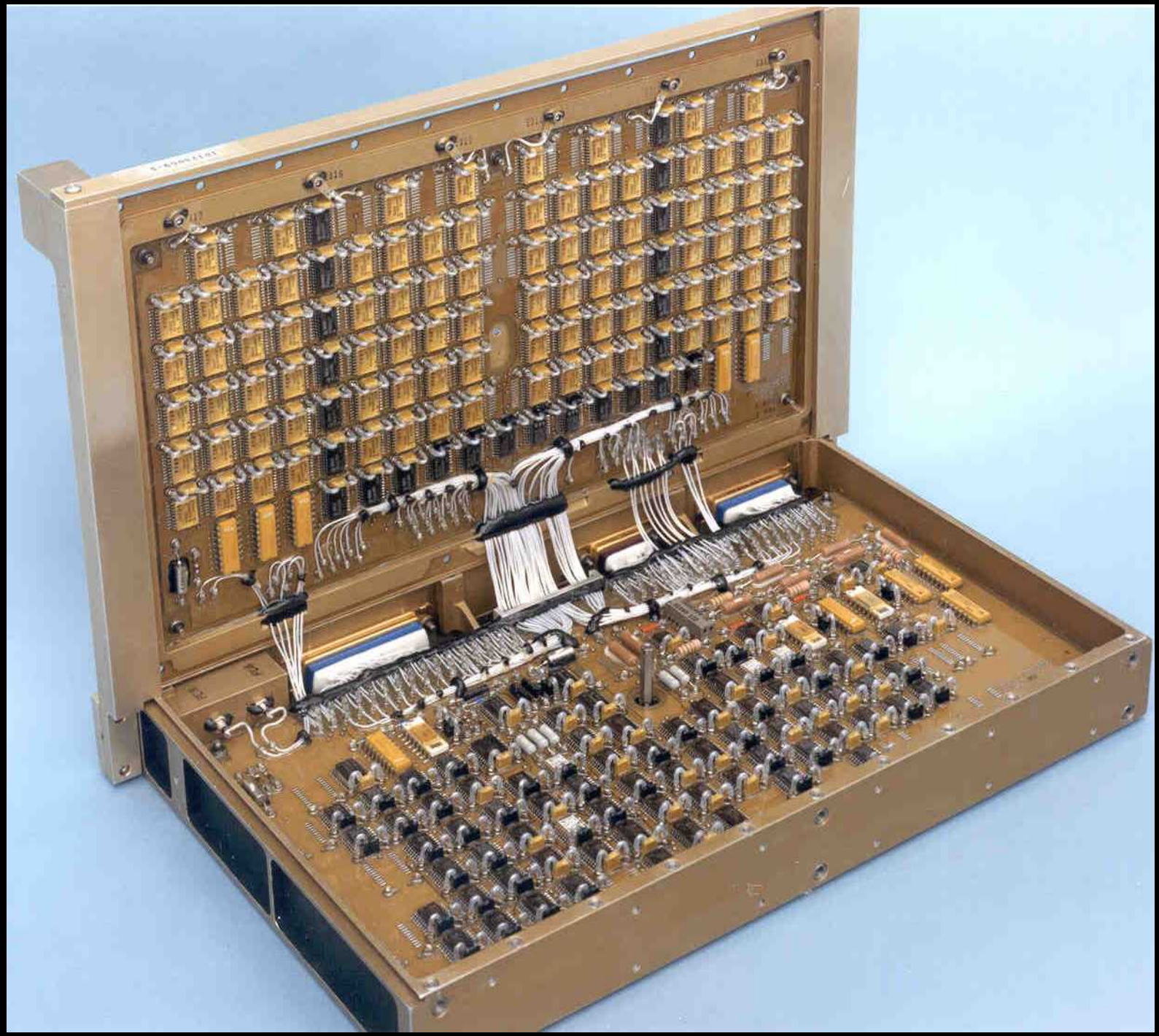
DESIGN ASSURANCE GUIDANCE FOR AIRBORNE ELECTRONIC HARDWARE, RTCA/DO-254

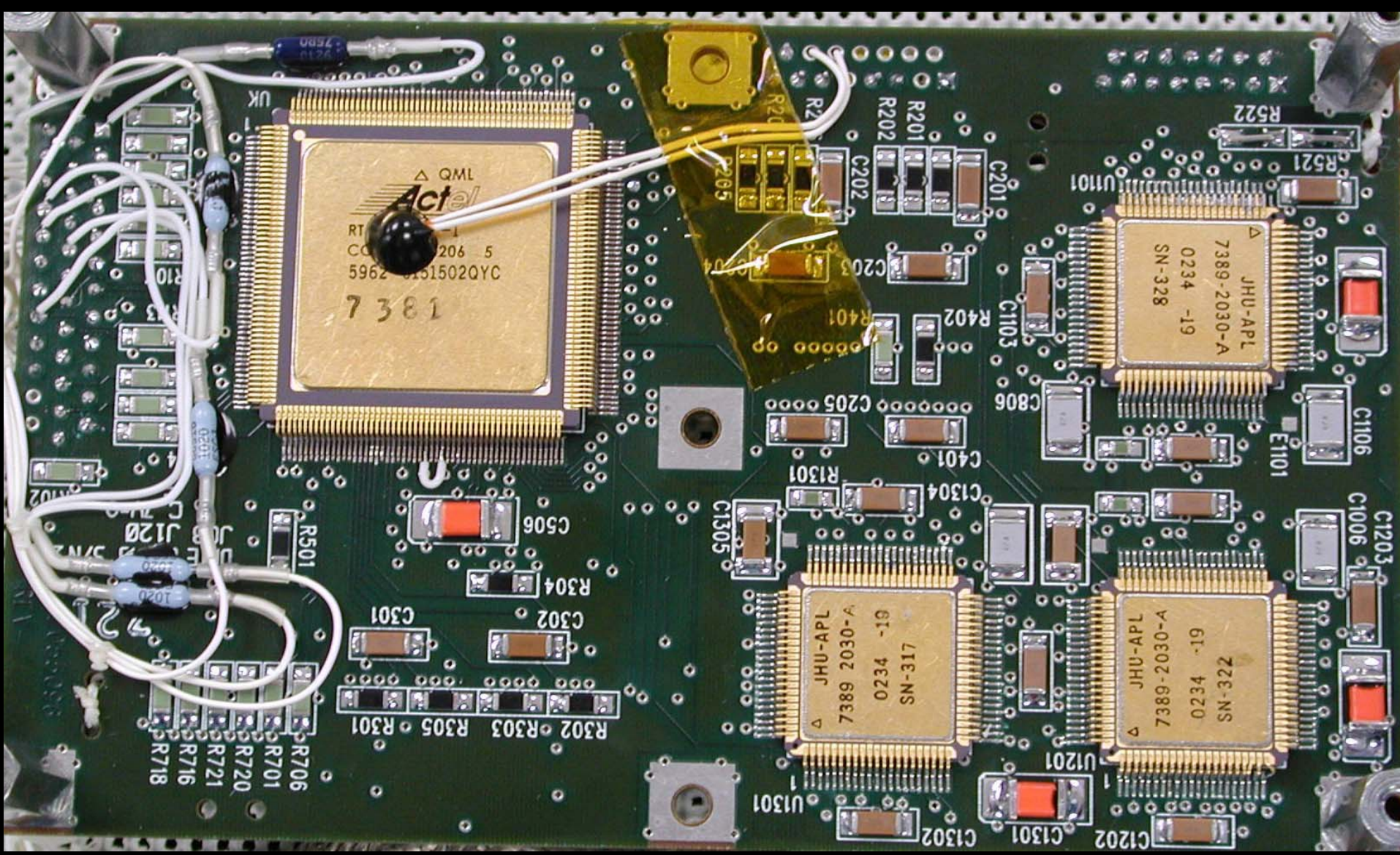
Definitions

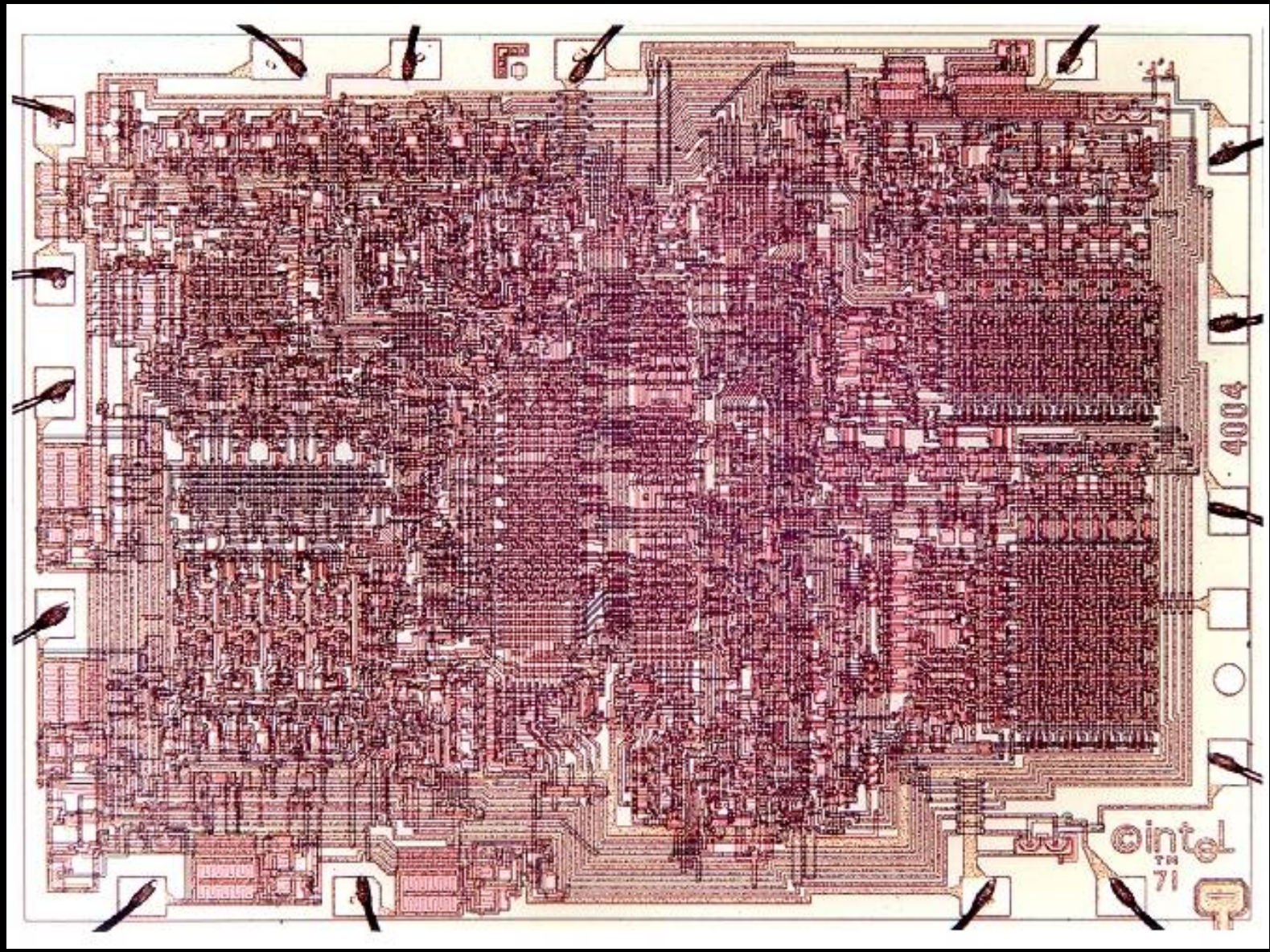
Complex Hardware Item *- All items that are not simple are considered to be 'complex'. See definition of Simple Hardware Item.*

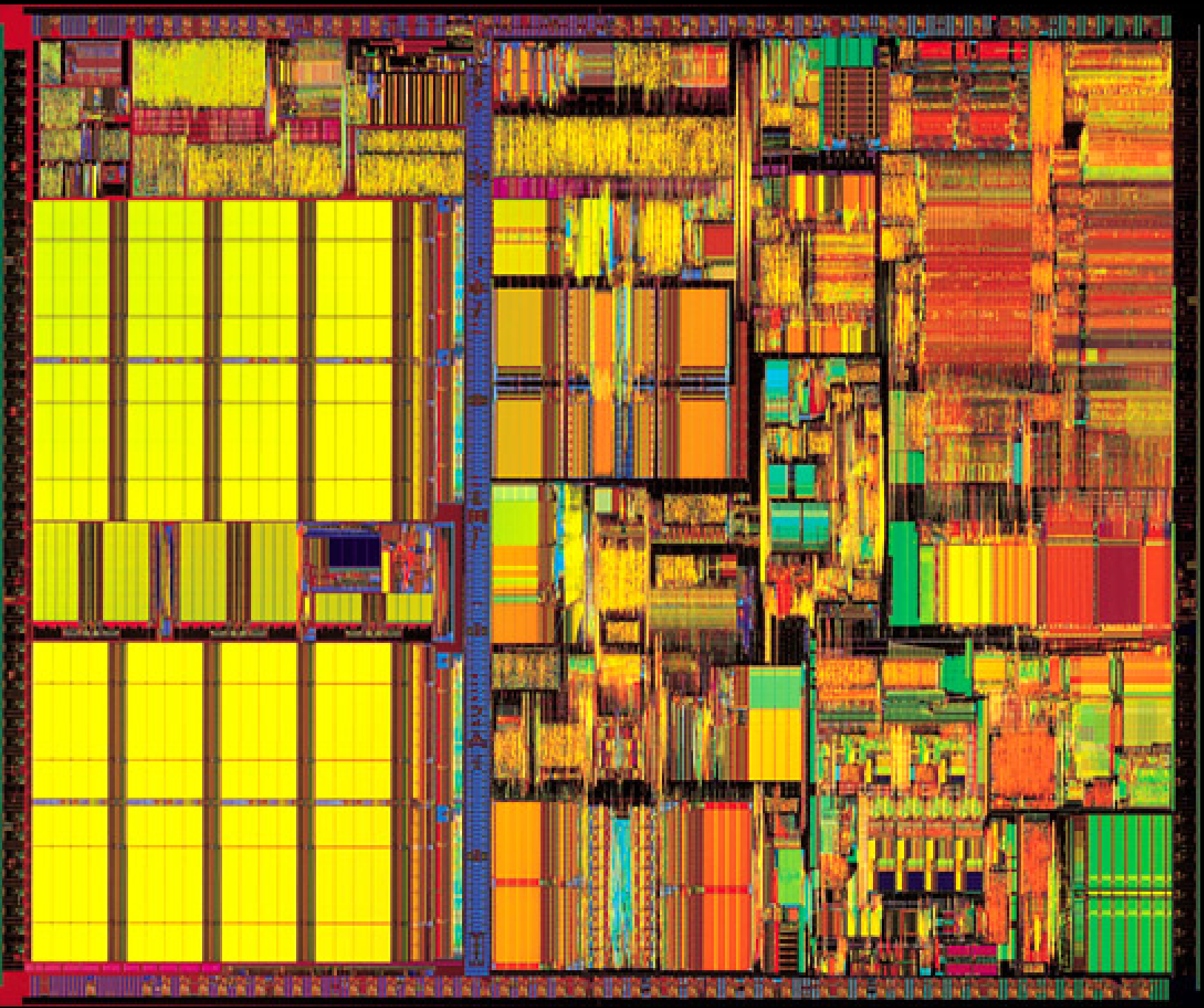


Simple Hardware Item *- A hardware item is considered simple if a comprehensive combination of deterministic tests and analyses can ensure correct functional performance under all foreseeable operating conditions with no anomalous behavior.*




QML
Actel
7381
JHU-APL
7389-2030-A
0234 -19
SN-328
JHU-APL
7389 2030-A
0234 -19
SN-317
JHU-APL
7389-2030-A
0234 -19
SN-322
U1101
U1201
U1301
U1
C201
C202
C203
C205
C401
C506
C1103
C1106
C1202
C1203
C1301
C1302
C1304
C1305
C1006
C806
R201
R202
R205
R401
R402
R501
R522
R521
R1301
R304
R302
R303
R305
R301
R718
R716
R721
R720
R701
R706
E1101
J120


4004
intel
TM
71



office of logic design

Is Hardware Software?

From DO-254

HDL design representations use coded text based techniques that are ***similar in appearance*** *to those used for software representations. This similarity in appearance can mislead one to attempt to use software verification methods directly on the design representation of HDL or other equivalent hardware specification languages.* ***The guidance of this document is applicable for design assurance for designs using an HDL representation.***

⇒ **This document covers HDL-based designs for safety critical aircraft functions**

DESIGN ASSURANCE GUIDANCE FOR AIRBORNE ELECTRONIC HARDWARE, RTCA/DO-254

Finite State Machine: Introduction

The Null Hypothesis:

Coding in HDLs can often lead to unreviewable designs, failures, and be slower, worse, and more expensive than schematics.

The Alternate Hypothesis

You guys are old and obsolete.

HDL = {VHDL. Verilog. C variant, Other HDL}

HDL Rationale

- Can design larger circuits
- Work at a higher level of abstraction
- The synthesizer does all the hard work
- The manufacturers will stop supporting schematic entry
- It's the "wave of the future"

A Flight VHDL Example (1)

```
-- signal declarations

--internal
signal  rst, cs, ale_n, pulse1_n, pulse2_n, gate, gate_del, int_tx_gate  : std_logic;
signal  ck8hz, det_start_count, det_stop_count : std_logic;

signal  datout  : std_logic_vector(7 downto 0);
signal  address  : std_logic_vector(7 downto 0);
signal  dec_addr  : std_logic_vector(5 downto 0);
signal  p1neg0, p1pos0, p2neg0, p2pos0  : std_logic_vector(19 downto 0);
signal  p1neg1, p1pos1, p2neg1, p2pos1  : std_logic_vector(19 downto 0);
signal  p1neg2, p1pos2, p2neg2, p2pos2  : std_logic_vector(19 downto 0);
signal  p1neg3, p1pos3, p2neg3, p2pos3  : std_logic_vector(19 downto 0);
signal  p1neg4, p1pos4, p2neg4, p2pos4  : std_logic_vector(19 downto 0);
signal  p1neg5, p1pos5, p2neg5, p2pos5  : std_logic_vector(19 downto 0);
signal  p1neg6, p1pos6, p2neg6, p2pos6  : std_logic_vector(19 downto 0);
signal  p1neg7, p1pos7, p2neg7, p2pos7  : std_logic_vector(19 downto 0);
signal  p1neg8, p1pos8, p2neg8, p2pos8  : std_logic_vector(19 downto 0);
signal  p1neg9, p1pos9, p2neg9, p2pos9  : std_logic_vector(19 downto 0);
signal  p1neg_tx, p1pos_tx  : std_logic_vector(19 downto 0);
signal  count : std_logic_vector(22 downto 0);
signal  latch_count : std_logic_vector(15 downto 0);
signal  start_count, stop_count : std_logic_vector(21 downto 0);
signal  count_pulses  : std_logic_vector(3 downto 0);
signal  reset_pulse_counter  : std_logic;
signal  stop_gate, reset_gate_pulse  : std_logic;

constant count_to_8hz: integer := 7812499;
```

A Flight VHDL Example (2)

```
begin
-- Component instances

power_on_reset  : DEMETA
   port map(ck => clk62_5mhz,
            reset_in => pwronrst(0),
            rstout => rst
);
counter_8hz : counter
          generic map
          (
                    num_bits => 23,
                    last_count => count_to_8hz
          )
port map(reset => rst,
         ck  => clk62_5mhz,
         enable => '1',
         count  => count
);

counter_pulses : counter
          generic map
          (
                    num_bits => 4,
                    last_count => 9
          )
port map(reset => reset_pulse_counter,
         ck  => pulse2_n,
         enable => '1',
         count  => count_pulses
);
```

A Flight VHDL Example (3)

```
ADDRESS_DECODER: decoder
          generic map
          (
                    innum_bits => 3,
                    outnum_bits => 6
          )

          port map
          (         din       => address(2 downto 0),
                    enable    => cs,
                    dec_addr  => dec_addr
          );

ADDRESS_LATCH  : reg
          generic map
          (
                    num_bits    => 8,
                    reset_value => 255
          )
  port map(data => ad(7 downto 0),
           ck => ale_n,
           reset => pwronrst(0),
           ena => '1',
           q => address
);
```

A Flight VHDL Example (4)

```
START_GATE_7_0   : reg
          generic map
          (
                    num_bits => 8,
                    reset_value => 255
          )
  port map(data => ad,
           ck => wr_n,
           reset => pwronrst(0),
           ena => dec_addr(0),
           q => start_count(7 downto 0)
);

START_GATE_15_8   : reg
          generic map
          (
                    num_bits
                    r
          )
  port
                                    (15 downto 8)
);
```

```
START_GATE_21_16   : reg
          generic map
          (
                    num_bits => 6,
                    reset_value => 63
          )
  port map(data => ad(5 downto 0),
           ck => wr_n,
           reset => pwronrst(0),
           ena => dec_addr(2),
           q => start_count(2
);

STOP_C
                                    lue => 250
                    ea => ad,
           ck => wr_n,
           reset => pwronrst(0),
           ena => dec_addr(3),
           q => stop_count(7 downto 0)
);
```

Yes, there are 16 pages of structural VHDL -- they were deleted for this presentation. This design was rejected and not flown.

"Computer Science World Mourning the Loss of Two of Its Trailblazers"

Dijkstra opposed the GOTO statement and worked to abolish it from programming. In a March 1968 letter to the editor of Communications of the ACM, **he contended that the more GOTO statements there are in a program, the harder it is to follow the source code.** The letter is acknowledged to have initiated the movement to produce reliable software by developing structured programs.

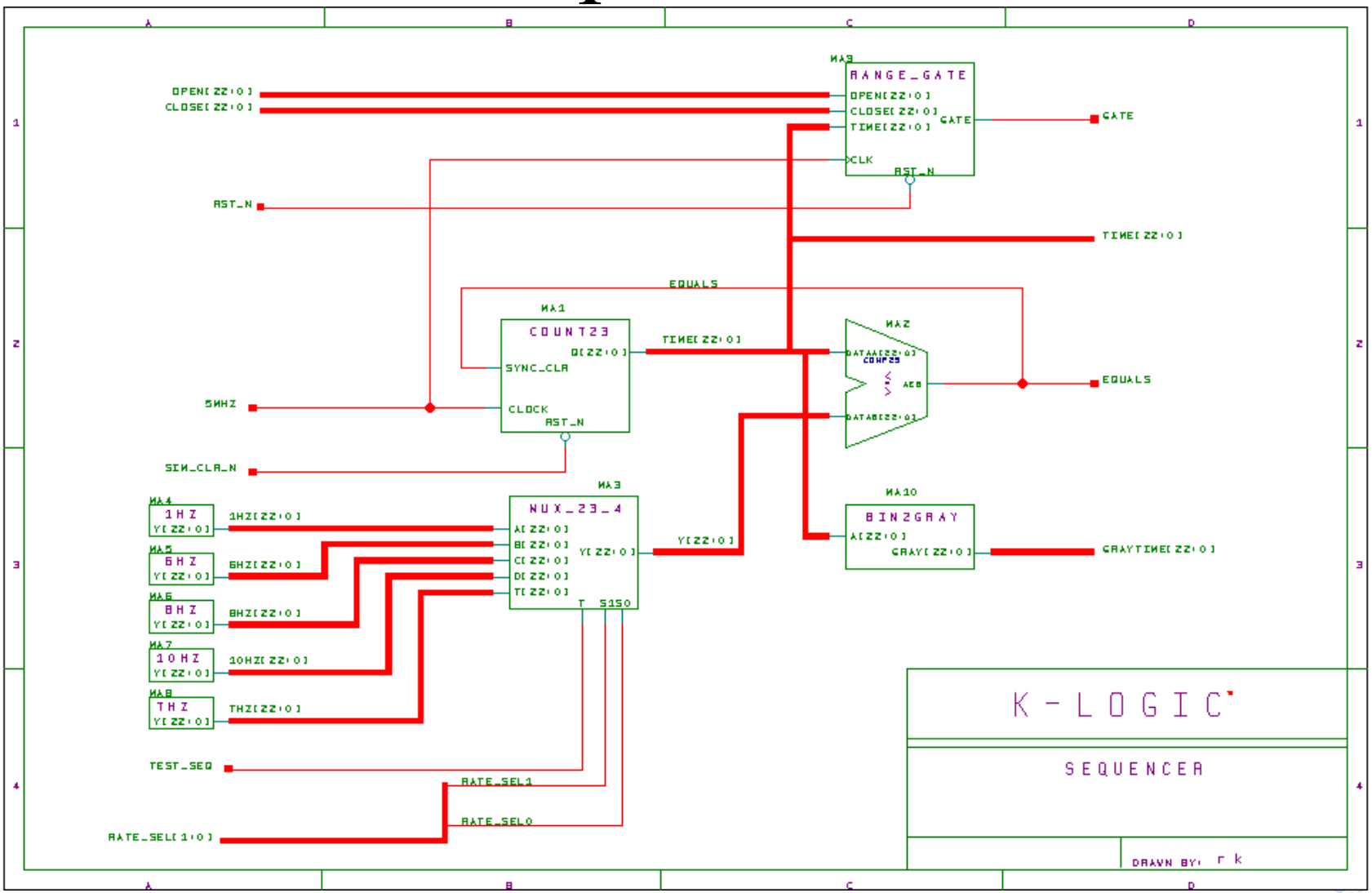
EE Times, August 26, 2002, p. 20

Think …

- What is the difference between a GOTO statement and structural VHDL?
- Why has the GOTO statement been considered **EVIL** by Computer Scientists for 40 years but is now present and prevalent in HDL-based hardware designs?

Finite State Machine Analysis

Ex: Master Sequencer, Schematic



Master Sequencer Example
VHDL Implementation[1]

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--
--
--
entity SEQUENCER is
    port( RST_N, C5MHZ, SIM_CLR_N  : in std_logic;
          THZ                      : in std_logic_vector(22 downto 0);
          TEST_SEQ                 : in std_logic;
          RATE_SEL                 : in std_logic_vector(1 downto 0);
          OPEN_VALUE,CLOSE_VALUE   : in std_logic_vector(22 downto 0);
          GATE                     : out std_logic;
          EQUALS                   : out std_logic;
          TIME_NOW, GRAYTIME       : out std_logic_vector(22 downto 0)
        );
end SEQUENCER;

architecture RTL_ARCH of SEQUENCER is
constant R1HZ  : std_logic_vector (22 downto 0) := std_logic_vector(to_unsigned(5000000,23));
constant R6HZ  : std_logic_vector (22 downto 0) := std_logic_vector(to_unsigned(833333,23));
constant R8HZ  : std_logic_vector (22 downto 0) := std_logic_vector(to_unsigned(625000,23));
constant R10HZ : std_logic_vector (22 downto 0) := std_logic_vector(to_unsigned(500000,23));
component COUNT23
    port( RST_N,SYNC_CLR, CLOCK : in std_logic;
          Q                     : out std_logic_vector(22 downto 0)) ;
end component;
component MUX_23_4
    port( A, B, C, D, T : in std_logic_vector(22 downto 0);
          T_SEL, S1, S0 : in std_logic;
          Y             : out std_logic_vector(22 downto 0)
        );
end component;
component COMP23
    port( DATAA, DATAB : in std_logic_vector(22 downto 0);
          AEB          : out std_logic
        );
end component;
component GATE_RANGE
    port( RST_N, CLOCK                    : in std_logic;
          OPEN_VALUE,CLOSE_VALUE, TIME_NOW : in std_logic_vector(22 downto 0);
          GATE                            : out std_logic
        );
end component;
component BIN2GRAY23
    port( A : in  std_logic_vector (22 downto 0);
          Y : out std_logic_vector (22 downto 0)
        );
end component;

signal Y : std_logic_vector (22 downto 0);
signal EQUALS_internal : std_logic;
signal TIME_NOW_internal : std_logic_vector(22 downto 0);

begin
 MA1: COUNT23  port map
              (RST_N => SIM_CLR_N, SYNC_CLR => EQUALS_internal, CLOCK => C5MHZ,
               Q => TIME_NOW_internal
              );
 MA3: MUX_23_4 port map
              (A => R1HZ, B => R6HZ, C => R8HZ, D => R10HZ, T => THZ,
               T_SEL => TEST_SEQ, S1 => RATE_SEL(1), S0 => RATE_SEL(0),
               Y => Y
              );
 MA2: COMP23   port map
              (DATAA => TIME_NOW_internal, DATAB=> Y,
               AEB => EQUALS_internal
              );
 MA7: GATE_RANGE port map
                (RST_N => RST_N,
                 CLOCK => C5MHZ,
                 OPEN_VALUE => OPEN_VALUE,CLOSE_VALUE => CLOSE_VALUE, TIME_NOW       => TIME_NOW_internal,
                 GATE => GATE
                );
 MA8: BIN2GRAY23 port map
                (A => TIME_NOW_internal,
                 Y => GRAYTIME
                );

 EQUALS <= EQUALS_internal;
 TIME_NOW <= TIME_NOW_internal;
end RTL_ARCH;
```

Broken down into sections and enlarged on the following pages.

[1] Design courtesy of an HDL design proponent.

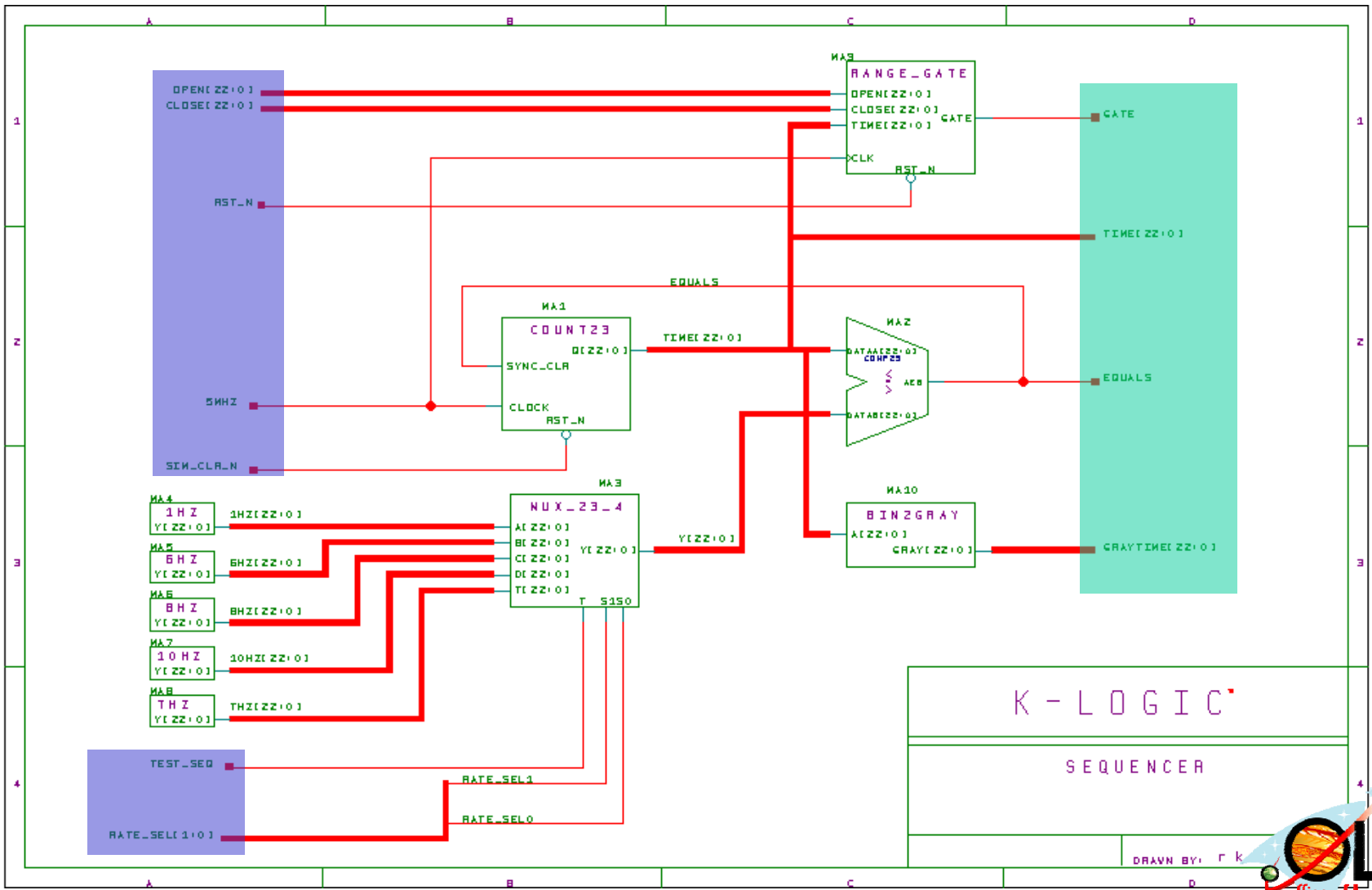
Header (e.g., symbol), constants, and various definitions deleted from this abridged version for brevity.

Master Sequencer Example
VHDL Implementation: Draw Nets

```
 MA1: COUNT23     port map
                  (RST_N => SIM_CLR_N, SYNC_CLR => EQUALS_internal, CLOCK => C5MHZ,
                   Q => TIME_NOW_internal
                  );
 MA3: MUX_23_4    port map
                  (A => R1HZ, B => R6HZ, C => R8HZ, D => R10HZ, T => THZ,
                   T_SEL => TEST_SEQ, S1 => RATE_SEL(1), S0 => RATE_SEL(0),
                   Y => Y
                  );
 MA2: COMP23      port map
                  (DATAA => TIME_NOW_internal, DATAB=> Y,
                   AEB   => EQUALS_internal
                  );
 MA7: GATE_RANGE port map
                  (RST_N => RST_N,
                   CLOCK => C5MHZ,
                   OPEN_VALUE => OPEN_VALUE,CLOSE_VALUE => CLOSE_VALUE,
                   TIME_NOW   => TIME_NOW_internal,
                   GATE => GATE
                  );
 MA8: BIN2GRAY23 port map
                  (A => TIME_NOW_internal,
                   Y => GRAYTIME
                  );

 EQUALS   <= EQUALS_internal;
 TIME_NOW <= TIME_NOW_internal;
end RTL_ARCH;
```

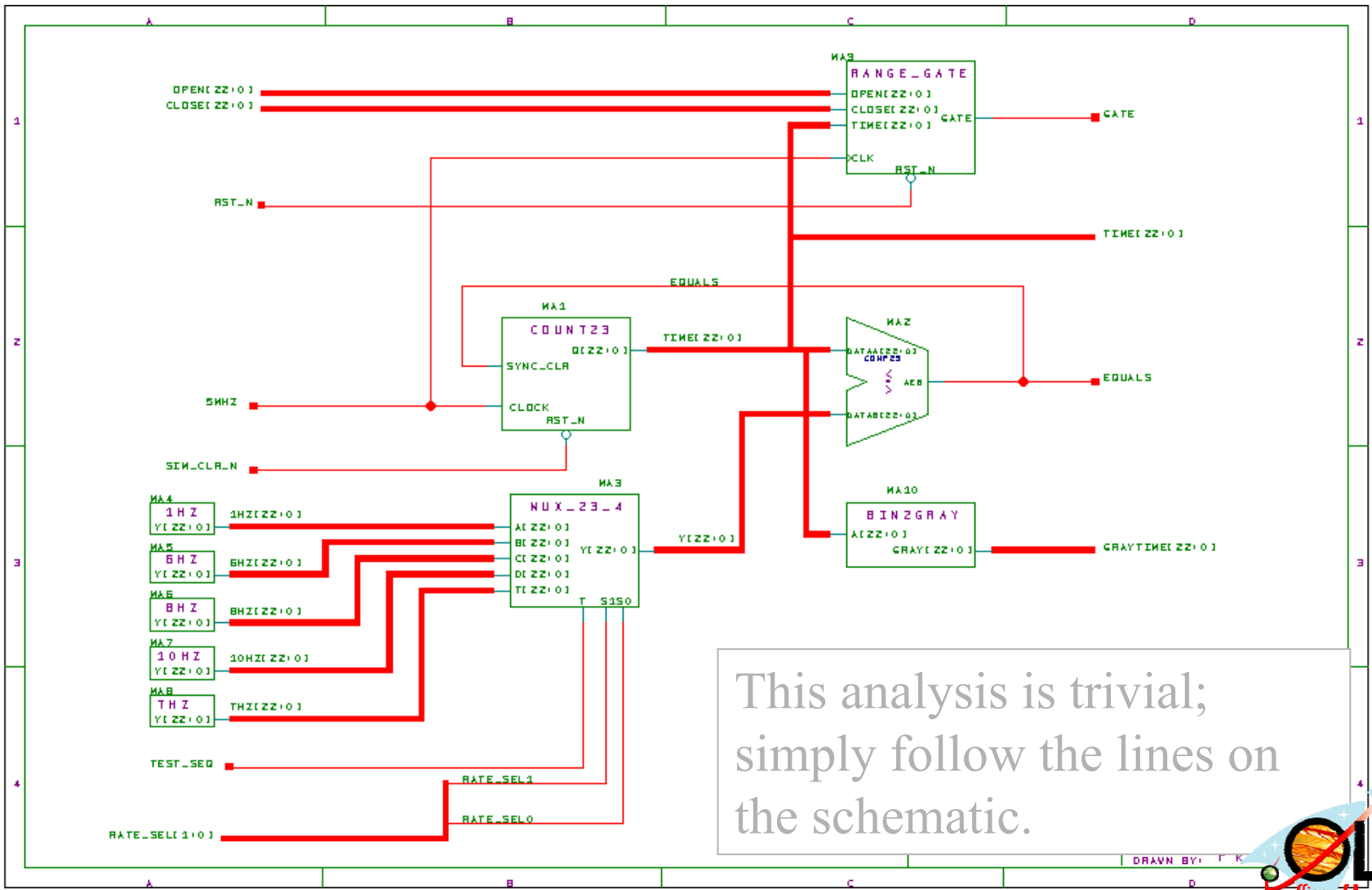
Master Sequencer Example

Schematic Analysis: Inputs and Outputs



Master Sequencer Example

Schematic Analysis: Connectivity



This analysis is trivial; simply follow the lines on the schematic.

Master Sequencer Example
Symbol Analysis: Inputs and Outputs

In designing this slide I put the colored boxes in the wrong spot. An error-prone methodology!

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--
--
--
entity SEQUENCER is
    port( RST_N, C5MHZ, SIM_CLR_N  : in std_logic;
          THZ                      : in std_logic_vector(22 downto 0);
          TEST_SEQ                 : in std_logic;
          RATE_SEL                 : in std_logic_vector(1 downto 0);
          OPEN_VALUE,CLOSE_VALUE   : in std_logic_vector(22 downto 0);
          GATE                     : out std_logic;
          EQUALS                   : out std_logic;
          TIME_NOW, GRAYTIME       : out std_logic_vector(22 downto 0)
        );
end SEQUENCER;
```

Master Sequencer Example

VHDL Analysis: Inputs and Outputs

```
MA1: COUNT23     port map
                 (RST_N => SIM_CLR_N, SYNC_CLR => EQUALS_internal, CLOCK => C5MHZ,
                  Q => TIME_NOW_internal
                 );
MA3: MUX_23_4    port map
                 (A => R1HZ, B => R6HZ, C => R8HZ, D => R10HZ, T => THZ,
                  T_SEL => TEST_SEQ, S1 => RATE_SEL(1), S0 => RATE_SEL(0),
                  Y => Y
                 );
MA2: COMP23      port map
                 (DATAA => TIME_NOW_internal, DATAB=> Y,
                  AEB   => EQUALS_internal
                 );
MA7: GATE_RANGE port map
                 (RST_N => RST_N,
                  CLOCK => C5MHZ,
                  OPEN_VALUE => OPEN_VALUE,CLOSE_VALUE => CLOSE_VALUE,
                  TIME_NOW   => TIME_NOW_internal,
                  GATE => GATE
                 );
MA8: BIN2GRAY23 port map
                 (A => TIME_NOW_internal,
                  Y => GRAYTIME
                 );

 EQUALS    <= EQUALS_internal;
 TIME_NOW <= TIME_NOW_internal;
end RTL_ARCH;
```

Note: Had to print out the entity just to make this slide.

Inputs

Outputs

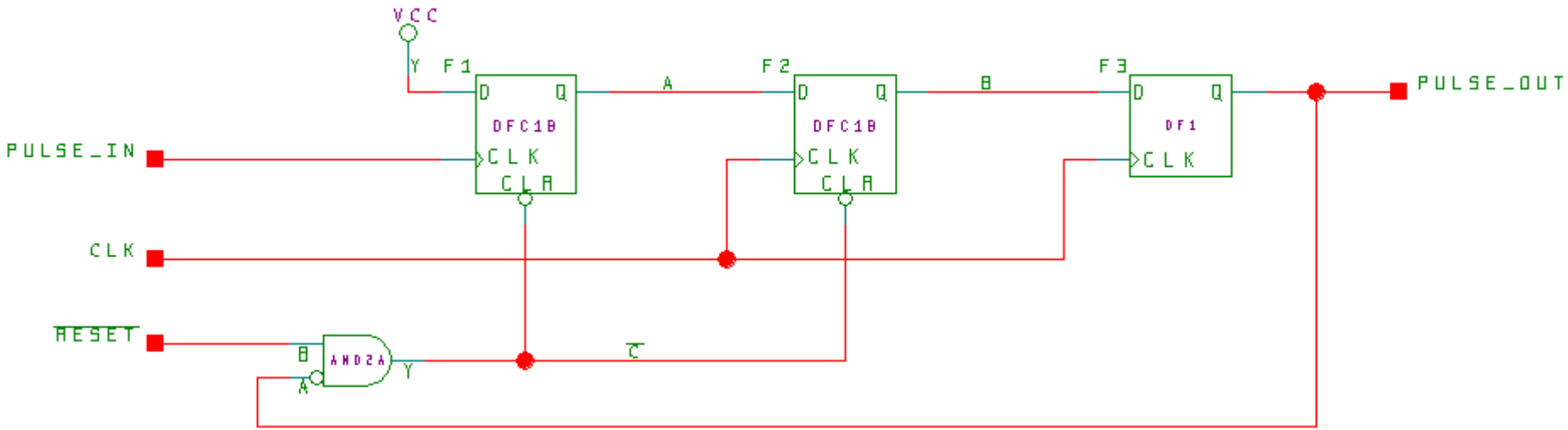
Master Sequencer Example

VHDL Analysis: Connectivity

```
MA1: COUNT23     port map
                 (RST_N => SIM_CLR_N, SYNC_CLR => EQUALS_internal, CLOCK => C5MHZ,
                  Q => TIME_NOW_internal
                 );
MA3: MUX_23_4    port map
                 (A => R1HZ, B => R6HZ, C => R8HZ, D => R10HZ, T => THZ,
                  T_SEL => TEST_SEQ, S1 => RATE_SEL(1), S0 => RATE_SEL(0),
                  Y => Y
                 );
MA2: COMP23      port map
                 (DATAA => TIME_NOW_internal, DATAB=> Y,
                  AEB   => EQUALS_internal
                 );
MA7: GATE_RANGE port map
                 (RST_N => RST_N,
                  CLOCK => C5MHZ,
                  OPEN_VALUE => OPEN_VALUE,CLOSE_VALUE => CLOSE_VALUE,
                  TIME_NOW   => TIME_NOW_internal,
                  GATE => GATE
                 );
MA8: BIN2GRAY23 port map
                 (A => TIME_NOW_internal,
                  Y => GRAYTIME
                 );

 EQUALS   <= EQUALS_internal;
 TIME_NOW <= TIME_NOW_internal;
end RTL_ARCH;
```

Making this chart was a lot of work and was error prone.

Simpler Example: Synchronizer Schematic Version



Gray Codes

"This is rookie stuff, so I can duck out of this module, get some cookies, and come back later, right?"

Reflected Gray and Binary Codes

- Single output changes at a time
  - Asynchronous sampling
  - Permits asynchronous combinational circuits to operate in fundamental mode
  - Potential for power savings
- Multiphase, multifrequency clock generator

	Binary	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0

VHDL Code for a 4-Bit Gray Code Sequencer

```
Library IEEE;
   Use IEEE.Std_Logic_1164.all;

Library Work;
   Use Work.Gray_Types.All;

Library synplify;
   Use synplify.attributes.all;

Entity Gray_Code Is

   Port ( Clock      : In  Std_Logic;
          Reset_N    : In  Std_Logic;
          Q          : Out States );
   End Entity Gray_Code;
```

```
Architecture RTL of Gray_Code Is
   Attribute syn_netlist_hierarchy of RTL : architecture is false;

   Signal IQ : States;
   Attribute syn_encoding of IQ : signal is "gray";

   Begin
     GC: Process ( Clock, Reset_N )
         Begin
           If ( Reset_N = '0' )
             Then IQ <= s0;
             Else If Rising_Edge ( Clock )
                    Then Case IQ Is
                           When s0     => IQ <= s1;
                           When s1     => IQ <= s2;
                           When s2     => IQ <= s3;
                           When s3     => IQ <= s4;
                           When s4     => IQ <= s5;
                           When s5     => IQ <= s6;
                           When s6     => IQ <= s7;
                           When s7     => IQ <= s8;
                           When s8     => IQ <= s9;
                           When s9     => IQ <= s10;
                           When s10    => IQ <= s11;
                           When s11    => IQ <= s12;
                           When s12    => IQ <= s13;
                           When s13    => IQ <= s14;
                           When s14    => IQ <= s15;
                           When s15    => IQ <= s0;
                           When Others => IQ <= s0;
                           End Case;
                      End If;
                  End If;
         End Process GC;

   Q <= IQ;

   End Architecture RTL;
```

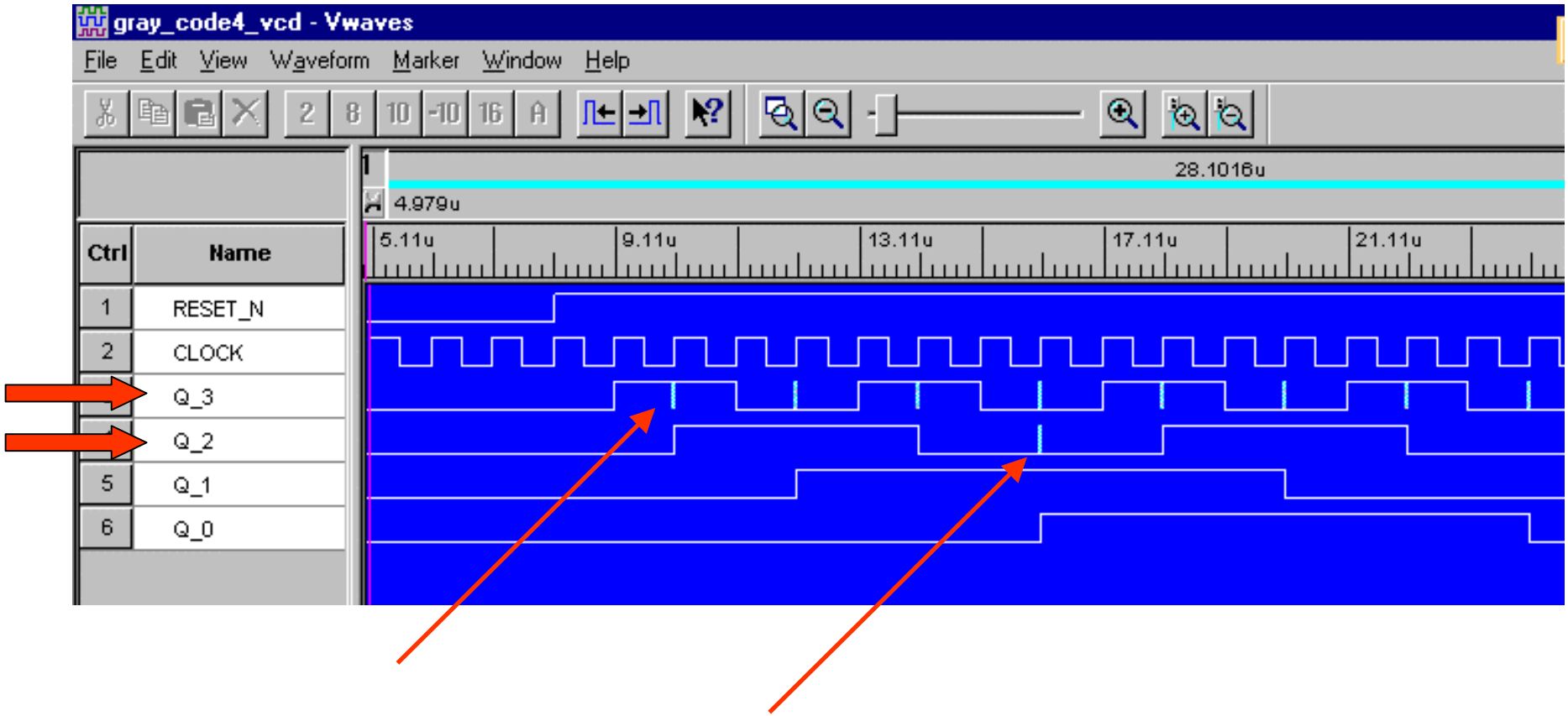
```
Package Gray_Types Is

   Type States Is ( s0,   s1,  s2,  s3,
                    s4,   s5,  s6,  s7,
                    s8,   s9, s10, s11,
                    s12, s13, s14, s15  );


   End Package Gray_Types;
```

Logic Simulation (1)

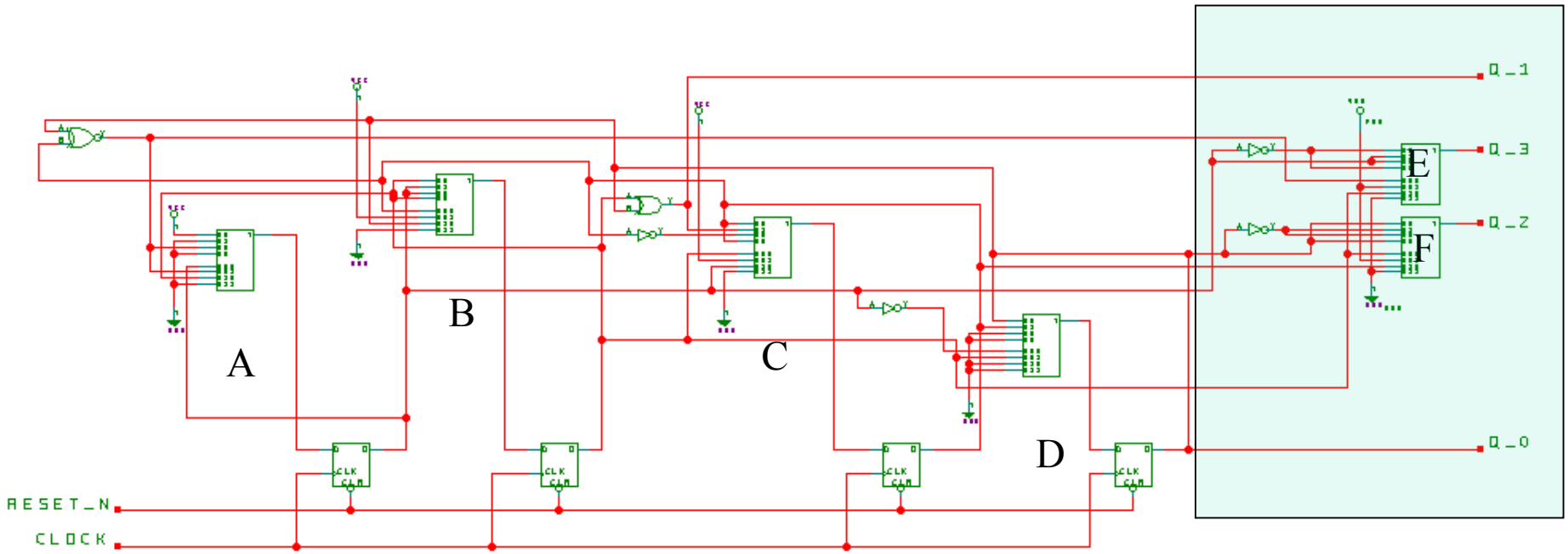
```
net -vsm "D:\designs\
sequencers\gray_code4.vsm"

clock clock 1 0
stepsize 500ns

vector q q_3 q_2 q_1 q_0
radix bin q
watch q

l reset_n
cycle 8

h reset_n
cycle 32
```

```
time =  9000.0ns  Q=0000
time = 10000.0ns  Q=1000
time = 11000.0ns  Q=1100
time = 12000.0ns  Q=0100
time = 13000.0ns  Q=0110
time = 14000.0ns  Q=1110
time = 15000.0ns  Q=1010
time = 16000.0ns  Q=0010
time = 17000.0ns  Q=0011
time = 18000.0ns  Q=1011
time = 19000.0ns  Q=1111
time = 20000.0ns  Q=0111
time = 21000.0ns  Q=0101
time = 22000.0ns  Q=1101
time = 23000.0ns  Q=1001
time = 24000.0ns  Q=0001
time = 25000.0ns  Q=0000
```

Logic Simulation (2)



Synthesizer Output for a Gray Code Sequencer



Logic Equations:

A: ~D2 ~S10+D2 ~S00

B: D0 ~S00 ~S10+D1 S00 ~S10+D1 ~S00 S10+D0 S00 S10

C: D0 ~S00 ~S10+D1 S00 ~S10+~D0 ~S00 S10+D0 S00 S10

D: D0 ~S01+D1 ~S00 S01+D0 S00

E: ~D0 ~S00 ~S10+D0 S00 ~S10+D0 ~S00 S10+~D0 S00 S10

F: D0 ~S00 ~S10 +~D0 S00 ~S10+~D0 ~S00 S10+D0 S00 S10

Outputs are not always driven by a flip-flop

Synthesis Issues

- Synthesizer ignored the command to make the state machine a Gray code and decided to make it a one-hot machine. Had to "fiddle" with the VHDL compiler settings for default FSM.
  - `Signal IQ : States;`
  - `Attribute syn_encoding of IQ : signal is "gray";`

- Output glitches!!!!!!!

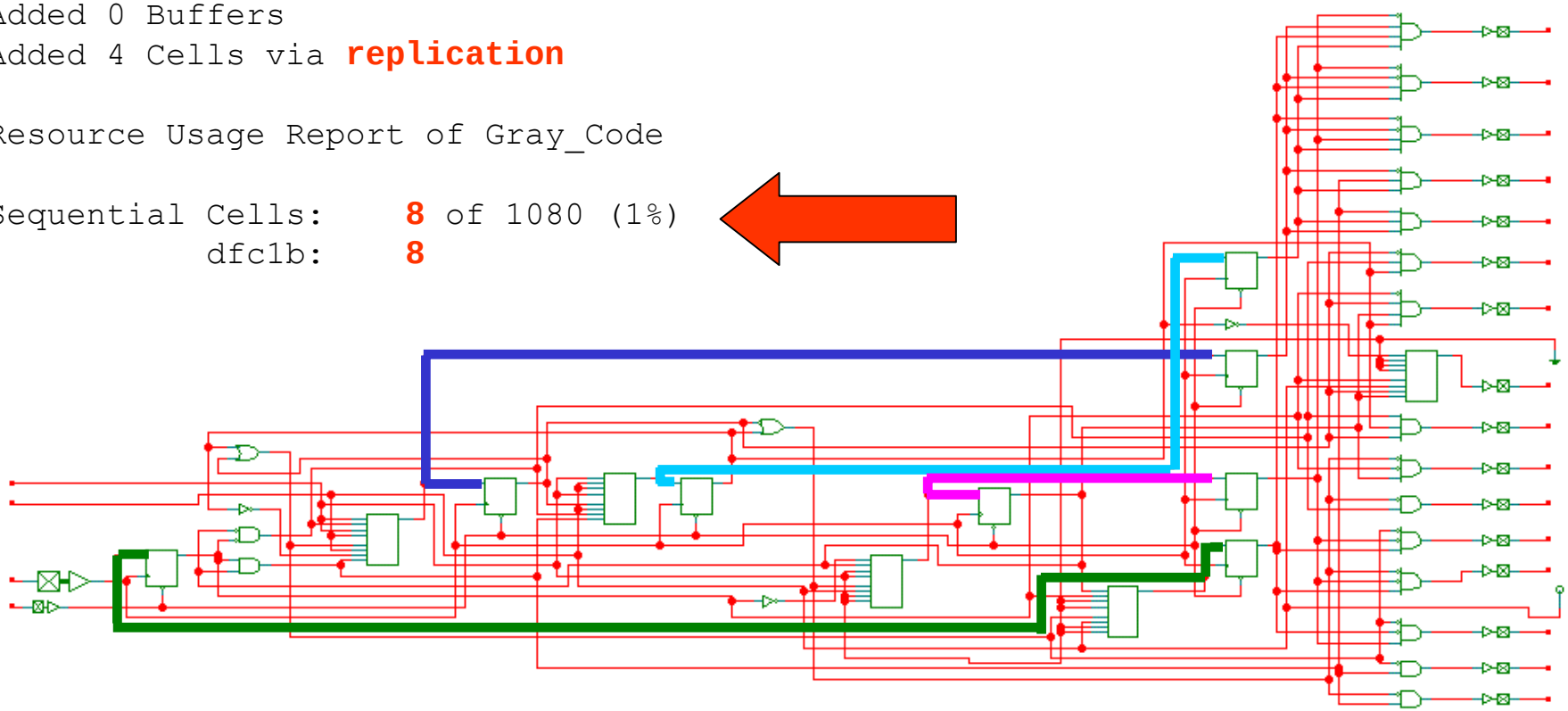
FSM Gray Codes and HDL The Saga Continues ...

We had another engineer (HDL specialist) run the same Gray coded FSM through his version of Synplicity and what did he get …

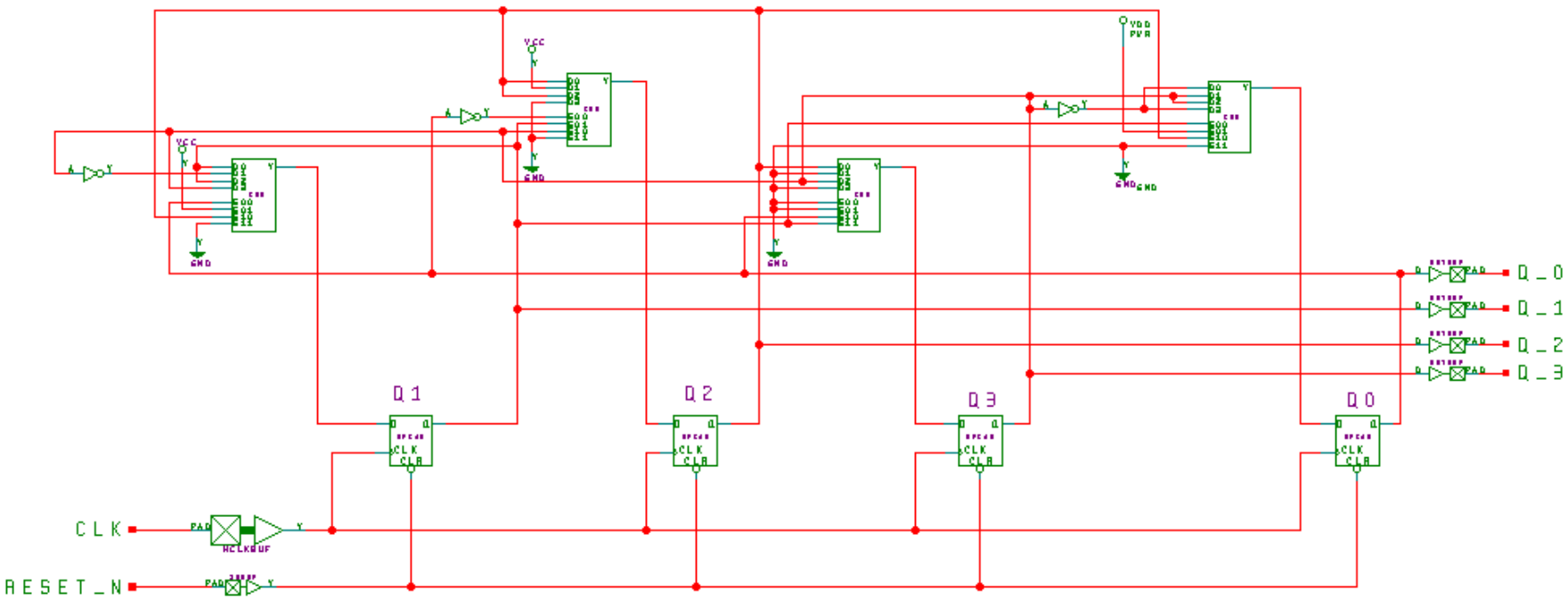
… Yes, as the cynic would expect, a different answer!

FSM Gray Codes and HDL
The Saga Continues ...

```
Replicating q_h.q[3], fanout 13 segments 2
Replicating q_h.q[2], fanout 13 segments 2
Replicating q_h.q[1], fanout 12 segments 2
Replicating q_h.q[0], fanout 12 segments 2

Added 0 Buffers
Added 4 Cells via replication

Resource Usage Report of Gray_Code

Sequential Cells:     8 of 1080 (1%)
           dfc1b:     8
```

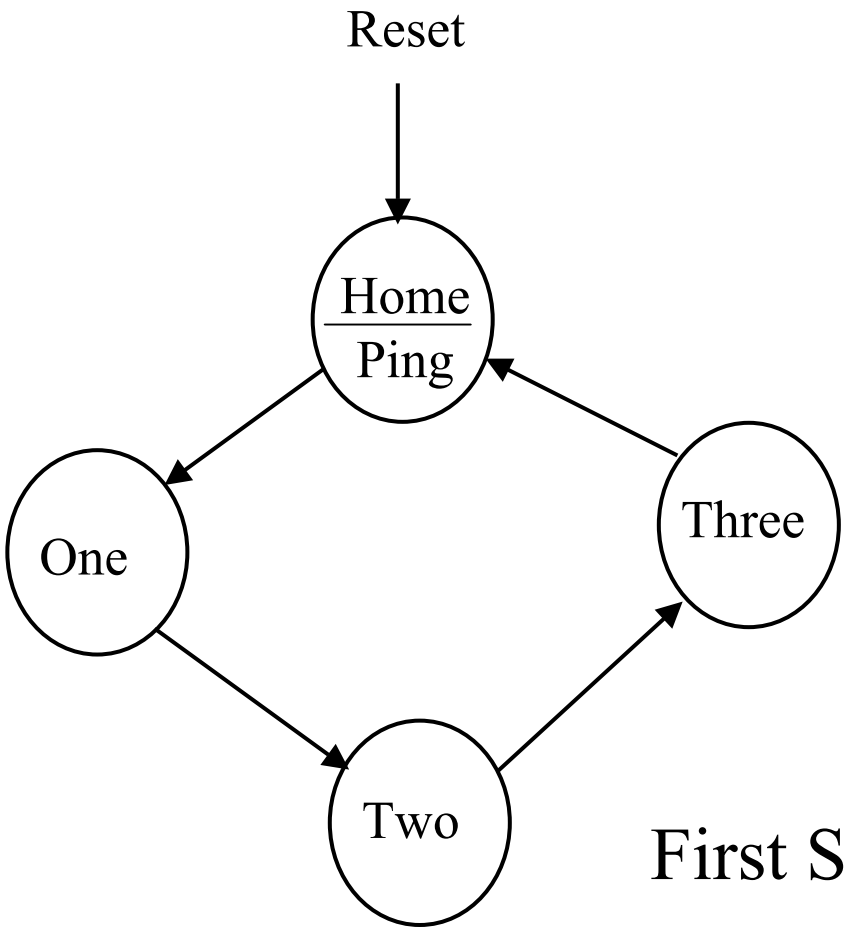


4-Bit Gray Code
No Enumerations or FSM Optimization



Flip-flop outputs routed directly to outputs.

Finite State Machines: Lockup States



First Sample State Machine

Finite State Machines

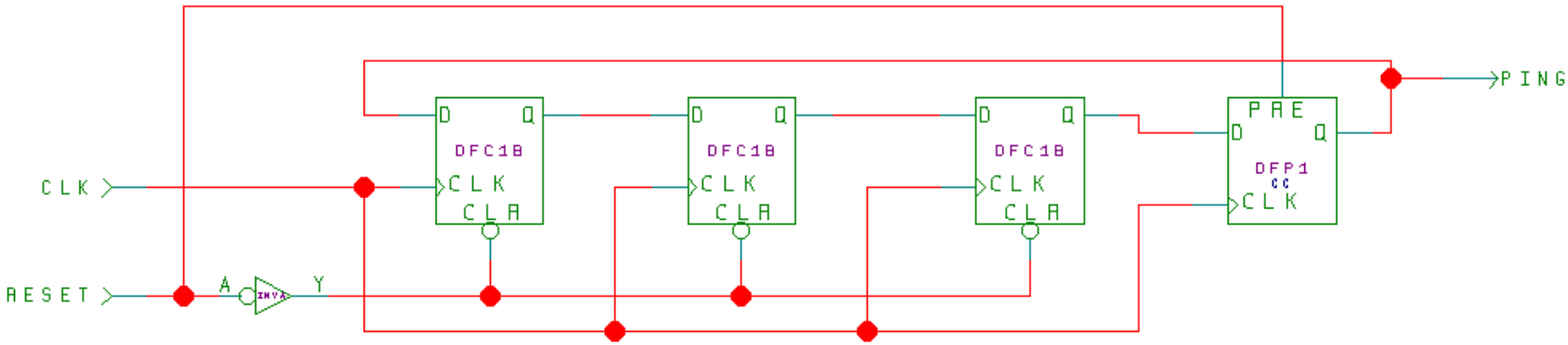
- Lockup State
  - A state or sequence of states outside the normal flow of the FSM that do not lead back to a legal state.
- CAE Tools - Synthesizers
  - Generates logic to implement a function, guided by the user.
  - Typically does not generate logic for either fault detection or correction, important for military and aerospace applications.

```
Library IEEE;  Use IEEE.Std_Logic_1164.All;
Entity Onehot_Simple_Act Is
  Port ( Clk   : In  Std_Logic;
         Reset : In  Std_Logic;
         Ping  : Out Std_Logic   );
End Onehot_Simple_Act;

Library IEEE;  Use IEEE.Std_Logic_1164.All;
Architecture Onehot_Simple_Act of Onehot_Simple_Act Is
Type      StateType Is ( Home, One, Two, Three );
Signal    State     :  Statetype;

Begin
  M: Process ( Clk, Reset )
     Begin
       If ( Reset = '1' )
         Then State <= Home;
         Else If Rising_Edge (Clk)
                Then Case State Is
                       When Home  => State <= One;
                       When One   => State <= Two;
                       When Two   => State <= Three;
                       When Three => State <= Home;
                       End Case;
                End If;
         End If;
     End Process M;
  O: Process (State)
     Begin
       If (State = Home)
         Then Ping <= '1';
         Else Ping <= '0';
         End If;
     End Process O;
End Onehot_Simple_Act;
```

Enumeration

Next-state Logic
All states “covered”

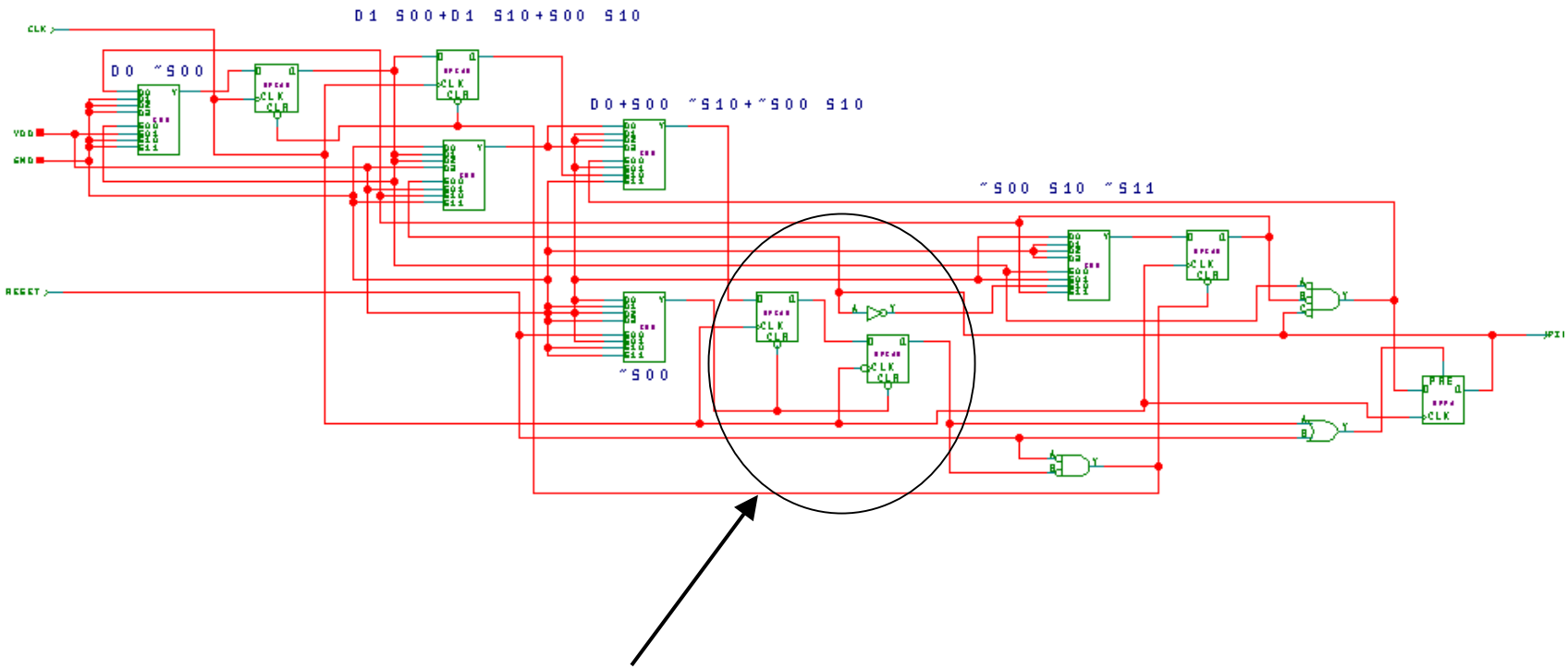
Lockup States
A Synthesized One-Hot Implementation



Typical ring counter with lockup states was synthesized.

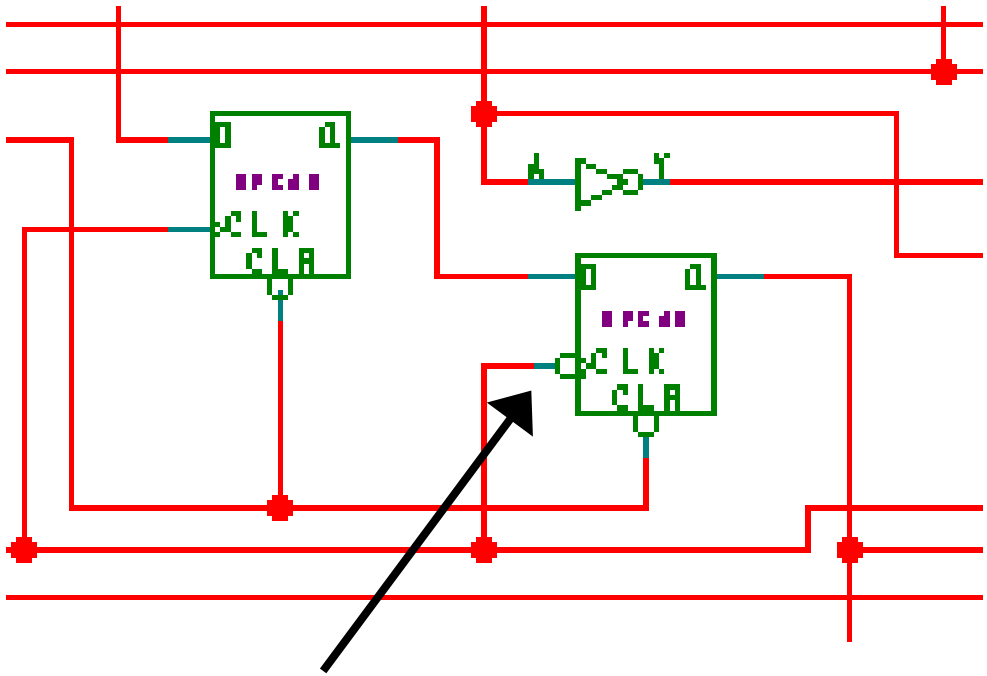
Note: The same synthesizer with slightly different inputs, versions, or constraints, can produce circuits with significantly different topologies and properties.

Lockup States
A "Safe" One-Hot Implementation (Synthesized)



Reset flip-flops. This implementation uses 6 flip-flops.

Lockup States
A "Safe" One-Hot Implementation (Synthesized)



Reset flip-flops. Note second one is on falling edge of the clock.

FSM Recommendations (Abridged)

- **Schematics for the Top Level of the Hierarchy:**
  - Graphical representation shows parallelism and pipelining.
  - Structural VHDL is similar (but worse!) then the GOTO in programming languages. In a programming language, targets of GOTOs are often easy to see graphically.
- **HDL for Components**
  - Components should normally be of the types that people can intuitively understand. VHDL code works well for many of these types, allowing reuse, customization of size, etc.
- **Complex State Machines**
  - VHDL also works well for expressing some complex state machines (CASE).
  - Ironically, for some simple sequential circuits, VHDL is ill-suited for the task.
- **Design for Reviews**
  - Designs that are too difficult to be reviewed will either not be reviewed well or at all.
- **Control Your Design**
- **Monitor Your Design**
  - Listing show number of f-fs, types, replication, state assignments, elimination of TMR, etc..
- **Verify your design thoroughly.**
  - Do not rely solely on simulation!!!!!
- **Look and think.**
  - Do not rely on these tools to do your thinking for you.

Complexity, System on a Chip, and Intellectual Property Issues

HDL allows designers to leave the confines of rigidly defined parts.

However, when constructing new components one shouldn't stray too far from the components that we all are familiar with from our board design days. There's a reason why those parts are the ones the manufacturers provided for many decades.

> In the same way that ANDs, ORs, and NOTs constitute a complete set for creating logic, higher level logic is made of things like muxes, counters, and decoders. While we may no longer be limited to choosing between 4-1, 8-1, and 16-1 muxes, we're still using muxes, everyone understands how they work, and there's probably some cosmic reason why the common MSI structures are the ones that have endured.

SoC Translation

OBC386 ➔ SoC

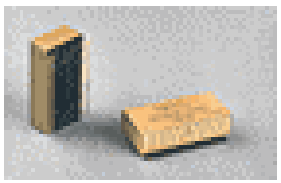
Bulk Data Storage (128MB)
IBM Microdrive

GLUE Logic
FPGA

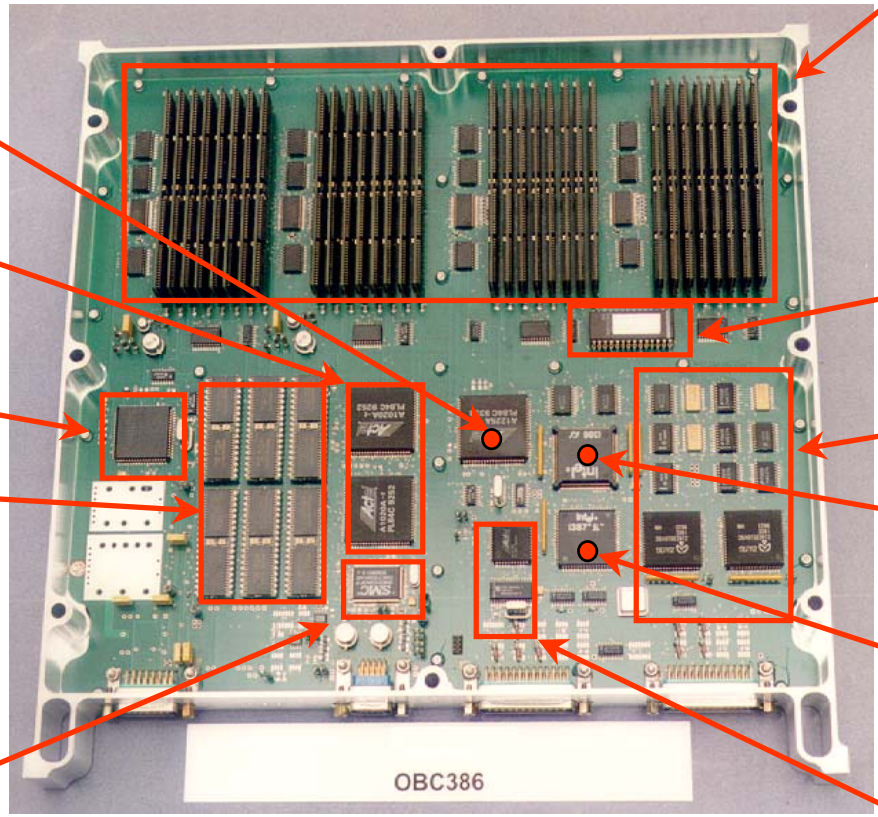
EDAC
SSTL IP Core

CAN TTC Node
ESA Hurricane IP Core

Program Memory (4MB)
3D+ Stackable SRAM



10BASE-2 Network
Bus LVDS



Reference Model

Bootloader (8KB)
FPGA LUT

HDLC Controller (4CH)
SSTL IP Core

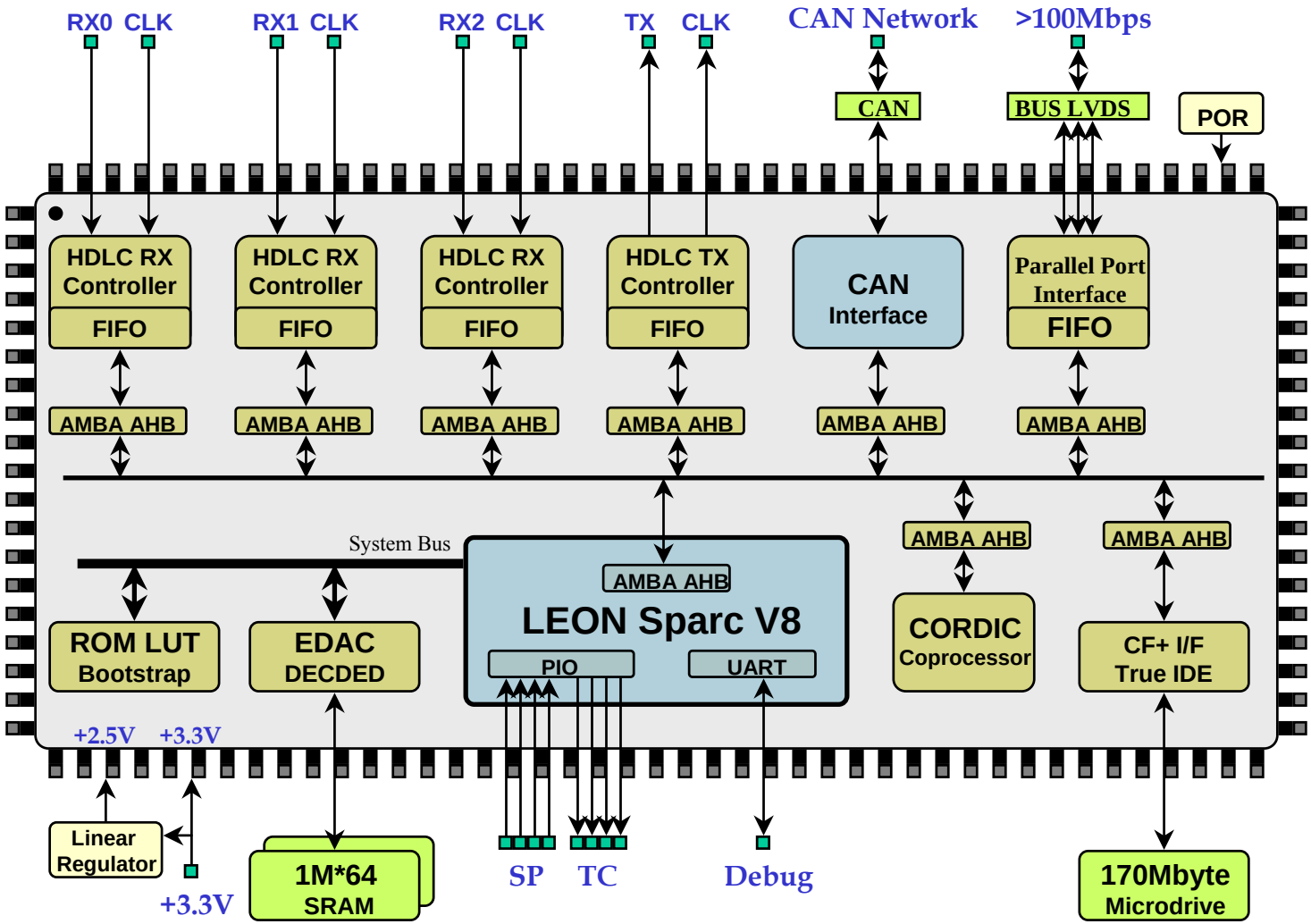
386EX Processor
ESA LEON IP Core

387SL CoProcessor
SSTL Cordic IP Core

CAN Peripheral
ESA Hurricane IP Core

A System-on-a-chip for Small Satellite Data Processing and Control
T. Vladimirova, H. Tiggeler, D. Zheng: Surrey Space Centre
2000 MAPLD International Conference

SoC Architecture



A System-on-a-chip for Small Satellite Data Processing and Control
T. Vladimirova, H. Tiggeler, D. Zheng: Surrey Space Centre
2000 MAPLD International Conference

Am29CPL154 Features

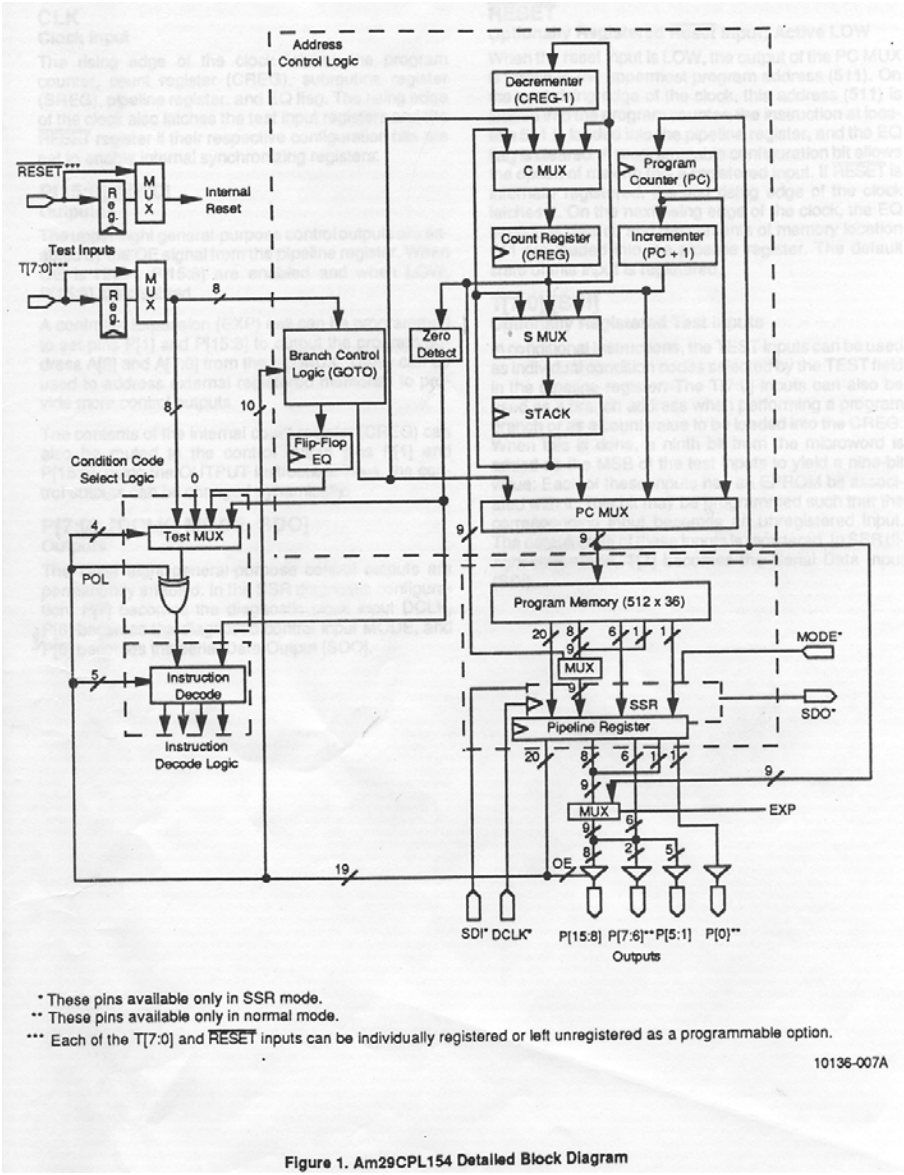
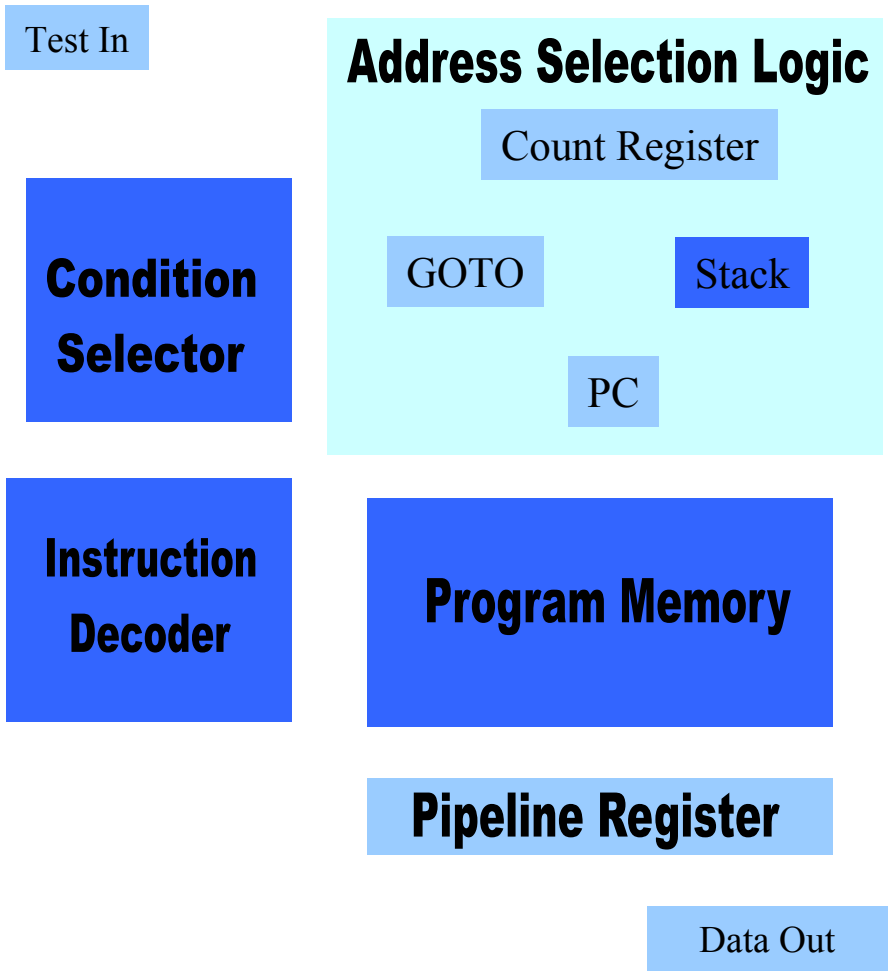


Figure 1. Am29CPL154 Detailed Block Diagram

- 512 x 36-bit Program ROM
- 8 test inputs, optionally registered
- 16 user outputs
- 28 instructions including conditional branching, looping, subroutine call, multiway branching
- 17-deep, 9-bit wide stack

"Reconfigurable, System-on-Chip, High-Speed Data Processing and Data Handling Electronics"
Igor Kleyner
1999 MAPLD International Conference
Laurel, Maryland

29KPL154 Capabilities



- Stack depth **selectable** from 0 to 17
- Program Memory internal, external or a combination
- Contents of **Program optimized** by Kompiler as ROM block
- **Instruction Encoding Optimized** by replacing unused fields with "don't_cares"
- Program contents analyzed by Kompiler, **only used instructions and test conditions implemented**
- 16 additional test inputs and 16 extra user outputs can be used

"Reconfigurable, System-on-Chip, High-Speed Data Processing and Data Handling Electronics"
Igor Kleyner
1999 MAPLD International Conference
Laurel, Maryland

Optimization Effort Concentrated In Darker Shaded Blocks

“Off the Shelf” IP: Some Issues

A recent look “under the hood” at some IP that is sold into the high-reliability space market showed:

- About the third party certification:
  - Was not tested at the frequency being sold at.
    - Code to support higher frequency was written after the certification.
  - Unknown whether the Verilog of VHDL version was tested.
- “Difficult” to run tests omitted.
- “Holes” in test bench.
- Finite state machines not analyzed for lockup states
- Logic design not analyzed for flip-flop replication.

CAE Tools and Models

CAE Tool Assessment and Qualification
(Mom and Apple Pie)

- *Tools, both hardware and software, will normally be used during hardware design and verification.*
  - *When design tools are used to generate the hardware item or the hardware design,* ***an error in the tool could introduce an error in the hardware item.***
  - *When verification tools are used to verify the hardware item,* ***an error in the tool may cause the tool to fail to detect an error in the hardware item or hardware design.***
- *Prior to the use of a tool, a tool assessment should be performed.*
- *The purpose of tool assessment and qualification is to ensure that the tool is capable of performing the particular design or verification activity to an acceptable level of confidence for which the tool will be used.*

From DO-254

CAE Tool Assessment and Qualification

From DO-254

- *Does the Tool have Relevant History? When it is possible to show that the tool has been previously used and has been found to produce acceptable results, then* ***no further assessment is necessary.***

How is an extrapolation of such a complex software product that is not open to inspection justified in a safety-critical application?

Isn't this just the Russian Roulette Theory of Aircraft Design?

CAE Tool Assessment and Qualification

From DO-254

- *"Basic Tool Qualification. Establish and execute a plan to confirm that the tool produces correct outputs for its intended application using analysis or testing. The tool's user guide or other description of the tool's function and its use may be used to generate requirements."*

1. **Does the user's guide/tool description fully and accurately define the tool for all cases?**
2. **Do vendors guarantee their tool's fidelity?**

A Logic Synthesizer: Manufacturer's View

Xxxxxxxxxx warrants that the program portion of **the SOFTWARE will perform substantially in accordance with the accompanying documentation** for a period of 90 days from the date of receipt.

IN NO EVENT SHALL XXXXXXXXXX OR ITS LICENSORS OR THEIR AGENTS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL OR INCIDENTAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTIONS, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF XXXXXXXXXX AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

A Simulator: Manufacturer's View

5. LIMITED WARRANTY.

5.1. Xxxxxx Xxxxxxxx warrants that during the warranty period Software, when properly installed, **will substantially conform to the functional specifications set forth in the applicable user manual**.

Xxxxxx Xxxxxxxx **does not warrant that Software will meet your requirements or that operation of Software will be uninterrupted or error free.**

XXXXXX XXXXXXXX AND ITS LICENSORS SPECIFICALLY **DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE** AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

Safety Critical Applications: Manufacturer's View

7. LIFE ENDANGERING ACTIVITIES. **NEITHER XXXXXX XXXXXXXX NOR ITS LICENSORS SHALL BE LIABLE** FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF SOFTWARE **IN ANY APPLICATION WHERE THE FAILURE OR INACCURACY OF THE SOFTWARE MIGHT RESULT IN DEATH OR PERSONAL INJURY.**

8. INDEMNIFICATION. **YOU AGREE TO INDEMNIFY AND HOLD HARMLESS** XXXXXX XXXXXXXX AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE, OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH YOUR USE OF SOFTWARE AS DESCRIBED IN SECTION 7.

Is proper qualification practical with complex CAE software?

Signal Integrity

Xxxxxxxx **makes no warranty** of its IBIS Model Software, expressed, implied, or statutory, including but not limited to warranties of merchantability and fitness for a particular purpose.

[IBIS ver] 3.2
[File name] XXXXX.ibs
[File Rev] 1.1
[Date] March 1, 2004
[**Disclaimer**] All V/I data was verified for accuracy against bench measurements. The measurements were done on **typical production parts**.
3.3V PCI model has not been verified against silicon measurements. Please check Xxxxx IBIS page for updates at http://www.xxxxx.com/

[IBIS ver] 2.1
[File name] XXXXXXX.ibs
[File Rev] 2.x
[Date] April 9, 2003
[Source] From Lab mesurement at Xxxxxxxx.
[**Disclaimer**] This information is for modeling purposes only, and is **not guaranteed**.

Timing Analysis

The findings below are accurate at the time of this posting and is the manufacturer's current guidance.

- **Minimum delay numbers calculated by the timing analysis tools are not guaranteed. They are not bound and actuals may be less then the reported values.**
- This is true for [static timing analyzer results] as well as files containing extracted delays such as .sdf files.
- …

When Should You and When Should You Not Use A Hardware Description Language (HDL)?

DO-254: "The guidance of this document is applicable for design assurance for **designs using an HDL representation**."

Some Typical Applications

- Critical Timing Circuit in a Scientific Instrument
  - Timing unit with < 400 ps resolution
- Controller for a Crane in an Industrial Environment.
  - Moving a Space Shuttle Orbiter
- Initiation Circuit for Explosives and Rockets
  - Warhead Fuzes
  - Self-Destruct Charges on a Solid Rocket Booster (manned)
  - Rocket Motor On Fighter Aircraft Missile

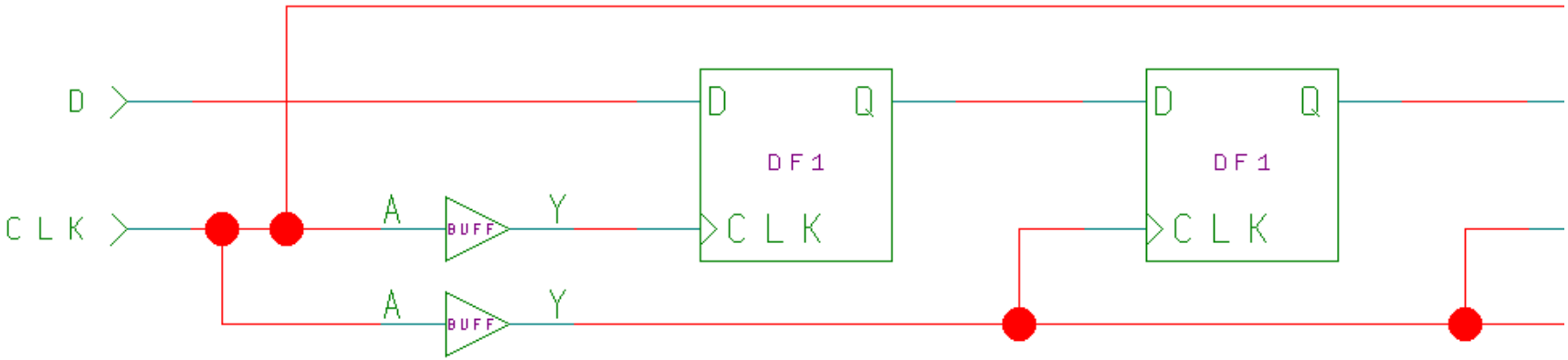
Critical Timing Circuit in a Scientific Instrument
Timing unit with < 400 ps resolution

- Don't have to like it, you just have to do it.
- Requires hand placement of many critical modules
  - Minimize Delays
  - Match Delays
    - Aid in calibration
    - Try to cancel temperature coefficients for t_{PD}
  - Assume on order of 100 modules must be hand placed.
- Schematic:
  - Straightforward to identify modules and place them. Names in the design match the names in the back end tool.
- VHDL:
  - Munges names, names constant from run to run? Effects on timing constraint/analysis tools?

A “Simple” Shift Register

```
Library IEEE;
  Use IEEE.Std_Logic_1164.All;

Entity Skew Is
  Port ( Clk   : In  Std_Logic;
         D     : In  Std_Logic;
         Q     : Out Std_Logic   );
End Skew;

Library IEEE;
  Use IEEE.Std_Logic_1164.All;

Architecture Skew of Skew Is
Signal ShiftReg : Std_Logic_Vector (31 DownTo 0);
Begin
  P: Process ( Clk )
     Begin
       If Rising_Edge (Clk)
         Then Q                       <= ShiftReg(0);
              ShiftReg (30 DownTo 0) <= ShiftReg (31 DownTo 1);
              ShiftReg (31)          <= D;
         End If;
     End Process P;
End Skew;
```

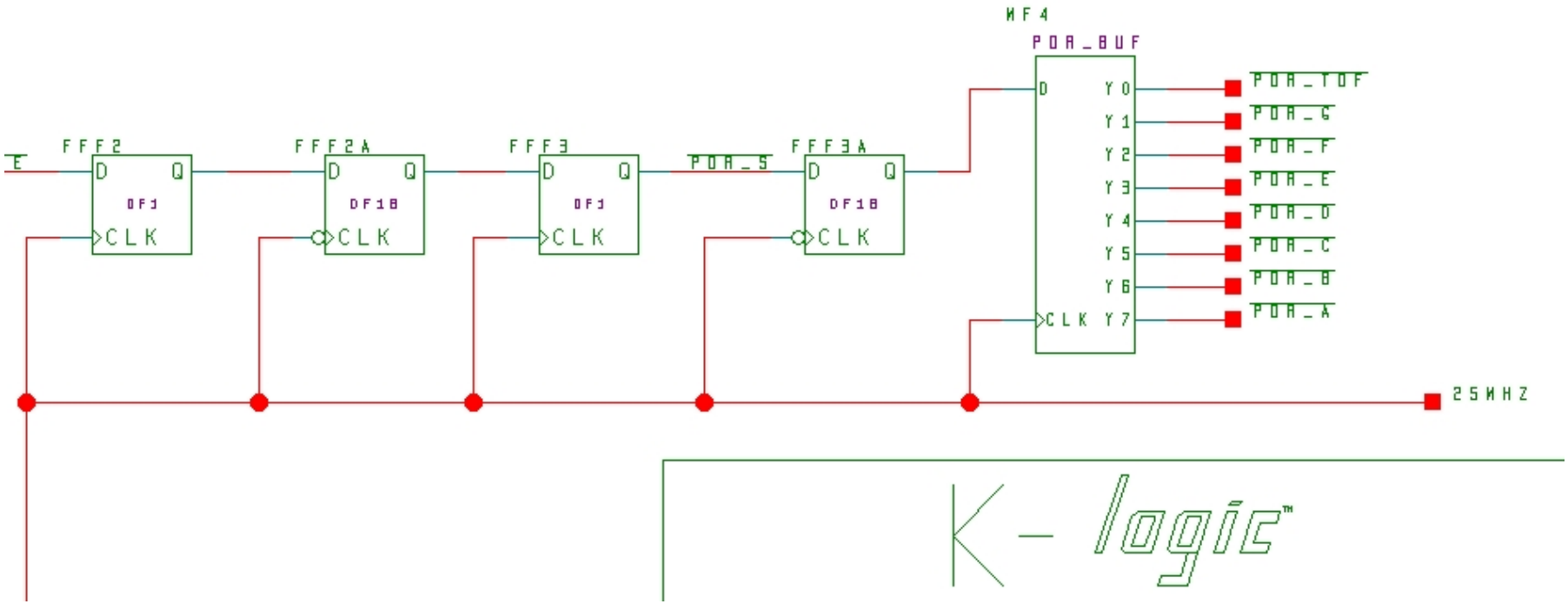
Clock Skew - From VHDL Synthesized Results



Results will depend on coding, directives and attributes, synthesizer, and synthesizer revision.

Here we see that the logic synthesizer generated a poor circuit.

A Schematic Approach to Skew-Tolerant Circuits



Opposite edge clocking technique

A VHDL Approach to Skew Tolerant Circuits
(Simplified Excerpt)

```
DTCountIntNEProc:
Process ( Clock, Reset )
Begin
  if Reset = ActiveReset then
    DTCountIntNE <= "00000000";
    elsif Falling_Edge ( Clock ) then
      if ReadPulse = '1' then
        DTCountIntNE <= DTCountInt + 1;
        end if;
    end if;
End Process DTCountIntNEProc;

DTCountIntProc:
Process ( Clock )
Begin
  if Rising_Edge ( Clock ) then
    DTCountInt <= DTCountIntNE;
    end if;
End Process DTCountIntProc;
```

Opposite edge clocking technique

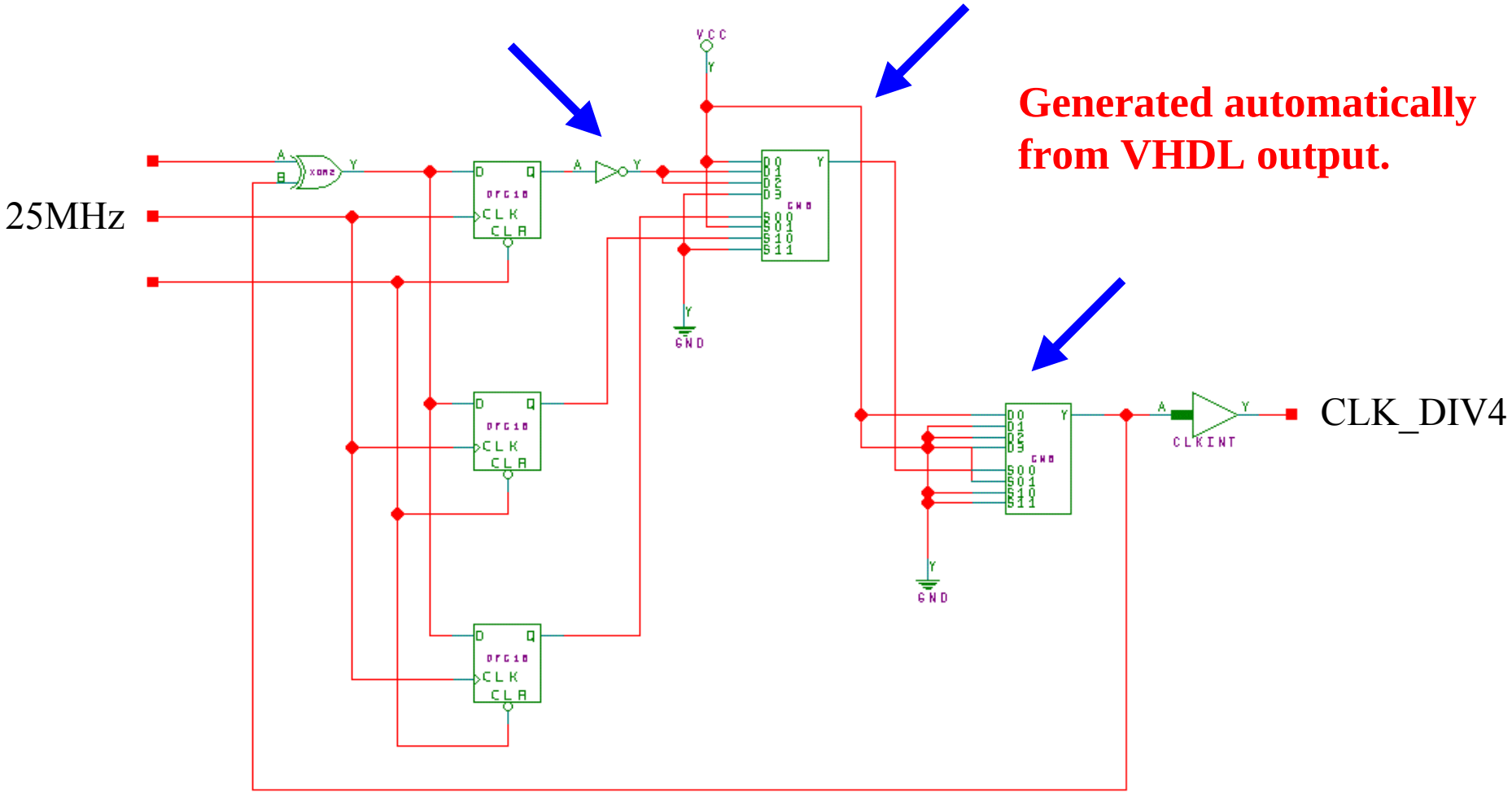
VHDL Code and Synthesizer Analysis

Case Study - Hardened Clock Generator

```
-- Divide 25 MHz (40 ns) clock by 4
-- to produce 6.25 MHz clock (160 ns)
-- This clock should be placed on
-- an internal global buffer

clkint1: clkint
Port Map ( A => clk_div_cnt(1),
           Y => clk_div4       );

clkdiv: Process (reset_n, clk)
Begin
  If reset_n = '0' Then
     clk_div_cnt <= "00";
  Elsif clk = '1' And clk'EVENT Then
          clk_div_cnt <= clk_div_cnt + 1;
  End If;
End Process clkdiv;
```

VHDL Code and Synthesizer Analysis

Case Study - Hardened Clock Generator



Most significant bit of the counter. 3 C-Cells are used for the voter. This circuit contains a hazard.

Language and Complexity

This is a simple example of language complexity

```
-- Age the debounced rxd signal to look for edges
                        ...
process (reset, sys_clk)
begin
  if reset = '1' then
    debounced_rx_last   <= '0';
    debounced_rx_change <= '0';
  elsif rising_edge(sys_clk) then
    debounced_rx_last <= debounced_rx;
    if debounced_rx_last /= debounced_rx then
      debounced_rx_change <= '1';
    else
      debounced_rx_change <= '0';
    end if;
  end if;
end process;
```

WYSIWYG

Switching from a schematic based to a synthesis based design involves several methodology changes. Some old tools are no longer needed while other newer tools must be learned. **A gate change is easier to do using a schematic.** An equivalent change made by an HDL and then synthesized may produce other changes since **the correspondence between the HDL and netlist produced as a result of synthesis can be obscure.**

" Design, Test, and Certification Issues for Complex Integrated Circuits," DOT/FAA/AR 95/31, L. Harrison and B. Landell, August 1996.

How Do **You** Verify Circuit Correctness for Safety Critical Applications?

Conclusions

3.2 Logic Design Pitfalls

The use of tools and **higher levels of design abstraction**, the pressures of rapid time to market (TTM), **inexperienced designers**, and other factors often contribute to designs of inferior quality. With the availability of more powerful design tools, **the actual logic implementations are further removed from the designer's critical inspection**. Designs often use libraries of functions which are supplied or purchased along with the tools. Both the tools and design libraries may contain design flaws that can escape the notice of designers. Higher levels of abstraction mean that **those who are not as familiar with digital design techniques and practices can now perform design functions.** What can suffer is the ability of the designer to verify that the circuit implemented by the tool suite is correct.

Those tasked with design verification should ensure that design pitfalls are avoided and that good design techniques are applied consistently.

" Design, Test, and Certification Issues for Complex Integrated Circuits," DOT/FAA/AR 95/31, L. Harrison and B. Landell, August 1996.

Conclusions: General

- **Barto's Law**: Every circuit is considered guilty until proven innocent.
- **Simple → Complex:** HDL's can make certain circuit structures more complex to design and verify.
  - Many vendors are now including schematic capability in their tool sets.

Conclusions: Simple and Complex Hardware

- Based on DO-254 Definitions:
  - "Simple Hardware" Is Not Easy
  - "Complex Hardware" Does Not Belong In Any Safety Critical Applications
- Size ≠ Complexity
- It's the engineer's job to manage increased design size and keep the complexity at practical levels. This is called subsystem engineering.

Conclusions: Simple and Complex Hardware

"These are highly complicated pieces of equipment almost as complicated as living organisms. In some cases, they've been designed by other computers. We don't know exactly how they work."

-- Scientist in Michael Crichton's 1973 movie, Westworld